UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Privacy in Cloud Computing

Francisco Emanuel Liberal Rocha

MESTRADO EM SEGURANÇA INFORMÁTICA

Dezembro 2010

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Privacy in Cloud Computing

Francisco Emanuel Liberal Rocha

Orientador
Miguel Nuno Dias Alves Pupo Correia

MESTRADO EM SEGURANÇA INFORMÁTICA

Dezembro 2010

Resumo

O paradigma *cloud computing* está progressivamente a integrar-se nas tecnologias de informação e é também visto por muitos como a próxima grande viragem na indústria da computação. A sua integração significa grandes alterações no modo como olhamos para a segurança dos dados de empresas que decidem confiar informação confidencial aos fornecedores de serviços *cloud*. Esta alteração implica um nível muito elevado de confiança no fornecedor do serviço. Ao mudar para a *cloud,* uma empresa relega para o fornecedor do serviço controlo sobre os seus dados, porque estes vão executar em hardware que é propriedade do fornecedor e sobre o qual a empresa não tem qualquer controlo. Este facto irá pesar muito na decisão, de mudar para a *cloud,* de empresas que tratam informação delicada (p.ex., informação médica ou financeira). Neste trabalho propomos demonstrar de que forma um administrador malicioso, com acesso ao hardware do fornecedor, consegue violar a privacidade dos dados que o utilizador da *cloud* confiou ao prestador desses serviços. Definimos como objectivo uma análise detalhada de estratégias de ataque que poderão ajudar um administrador malicioso a quebrar a privacidade de clientes da *cloud*, bem como a eficácia demonstrada contra esses mesmos ataques por mecanismos de protecção já propostos para a *cloud*. Pretendemos que este trabalho seja capaz de alertar a comunidade científica para a gravidade dos problemas de segurança que actualmente existem na *cloud* e, que ao mesmo tempo, sirva como motivação para uma acção célere desta, de forma a encontrar soluções para esses problemas.

Palavras-chave: cloud computing, segurança, confidencialidade, privacidade.

Abstract

The paradigm of cloud computing is progressively integrating itself in the Information Technology industry and it is also seen by many experts as the next big shift in this industry. This integration implies considerable alterations in the security schemes used to ensure that the privacy of confidential information, companies entrust to the cloud provider, is kept. It also means that the level of trust in the cloud provider must be considerably high. When moving to the cloud, a company relinquishes control over its data to the cloud provider. This happens because, when operating in the cloud, the data is going to execute on top of the hardware owned by the cloud provider and, in this scenario, the client has no control over that hardware. Companies that deal with sensitive data (e.g., medical or financial records) have to weigh the importance of this problem when considering moving their data to the cloud. In this work, we provide a demonstration of how a malicious administrator, with access to the hardware of the cloud provider, is capable of violating the privacy of the data entrusted to the cloud provider by his clients. Our objective is to offer a detailed analysis of attack strategies that can be used by a malicious administrator to break the privacy of cloud clients, as well as the level of efficacy demonstrated by some protection mechanism that have already been proposed for the cloud. We also hope that this work is capable of capturing the attention of the research community to the security problems existent in the cloud and, that at the same time, it works as a motivation factor for a prompt action in order to find solutions for these problems.

Keywords: cloud computing, security, confidentiality, privacy.

Acknowledgments

This project could not have reached this final outcome without the crucial help of Professor Miguel Pupo Correia, my advisor. That is why I would like to thank him for the excellent guidance he gave me throughout all the stages of my graduate project.

I would also like to thank my family and friends for always supporting my decisions and being there for me when I needed the most.
All this would not make any sense without all of you, thank you.

Lisboa, December 2010

Dedicated to my family and friends.

Contents

List of Figures

Chapter 1

Introduction

Cloud computing is currently a considerably active area in the computer science field. It is safe to say it due to the amount of scientific research done on the subject and the quantity of new products using the cloud as their foundation. Throughout this document we will discuss some of the published research papers related to the topic, and mention some of the products that are already taking advantage of cloud computing technology. Since the paradigm of Cloud computing seems to be getting a large acceptance, we believe it is in the best interest of future Cloud users to learn about the security issues they might face when deciding to move their businesses to the Cloud. The benefits of moving to the Cloud have already been discussed and enumerated in multiple texts [1, 2]. We present a discussion on this particular subject in Section 2.1. The question that we still need to answer is whether or not high-value businesses can move to the cloud when the security risks they face are almost prohibitive. Showing that these risks are serious is what we propose to achieve with this work. We are going to perform an analyses demonstrating to current cloud clients, how insecure is their data going to be once they relinquish its control over to the Cloud provider. We also hope that our work can serve as a call for action directed at the research community, which will hopefully solve the problems we present here. In the thesis we take the word privacy in a broad sense, including any information that a company or a cloud user considers to be confidential or private.

Our main contribution is the demonstration of a set of attacks showing that there are considerable risks worth pondering about before deciding to move high-value businesses into the Cloud. We present them from the perspective of a malicious insider that is located within the cloud infrastructure premises, for instance a system administrator that goes rogue. To the best of our knowledge this perspective has not been previously ventured into in the research community. Recent work [29] has made it public that it is quite easy to obtain memory dumps from machines and then use that data to extract confidential information,

e.g., private keys, that was loaded in the memory of the victim machine. We explore this family of attacks and others applied to the Cloud environment. Our findings are definitely interesting and will certainly foment the discussion on how secure it is to move to currently available Cloud solutions.

The document is divided in three main chapters, conclusions and an appendix. The first of the main chapters is composed by context and related work to Cloud computing. It has the objective of providing an introduction to the subjects of cloud computing, virtualization and trusted computing. The content provided in this chapter is vital for the understanding of the remainder of this document. Chapter 3, the second from the main chapters, is dedicated to exploiting privacy issues in cloud computing. In this part of the document we present possible attacks that enable a malicious administrator to compromise the privacy of data that belongs to the Cloud user. This is the most important chapter of this document since it is in it we present the reader with proofs and demonstrations that, in current cloud infrastructures, it is indeed possible to violate the privacy of the Cloud user. The last chapter of this series of three is dedicated to some protection mechanisms that have already been proposed to solve the security challenges faced in the Cloud environment. Some of these mechanisms are proposals of solutions that try to solve the whole problem, and others are individual mechanisms that can be used in building this type of solutions. We show that none of them is sufficient to solve the problem. The remainder of the text is dedicated to our conclusions, some future work we consider relevant and an appendix that is no more than a tutorial to reproduce our testing environment.

Chapter 2

Context and Related Work

The purpose of this chapter is to present a background overview of relevant topics that serve as foundations for the cloud computing paradigm or are potential tools to make it secure. However, the intention is not to provide a thorough description of each subject but instead a general overview focusing in the most important subtopics, so the reader can follow the text.

This chapter contains three main sections. In Section 2.1 we provide an introduction of the notions related to Cloud computing. The security of Cloud Computing is discussed in Section 2.2. Section 2.3 introduces the concepts of virtualization that are relevant to understanding this material, and it also briefly presents the architecture of the Xen hypervisor in Subsection 2.3.3. The chapter closes with Section 2.4, which we have dedicated to the properties offered by trusted computing.

2.1 Cloud Computing

> *"It's one of the foundations of the next generation of computing. . . . It's a world where the network is the platform for all computing, where everything we think of as a computer today is just a device that connects to the big computer we are building. Cloud computing is a great way to think about how we will deliver computing services in the future."*
>
> *- Tim O'Reilly, CEO, O'Reilly Media*

The concepts behind cloud computing date back to the 60s [54], but only recent efforts from Amazon, Google and IBM brought the notion of cloud computing to the attention of industry and academia. Virtualization, which we will present later with some detail, is considered

to be the cornerstone technology for this revival. It is said so because virtualization is the technology that allows the abstraction of computer resources and this makes it possible to make a better use of available computing resources. So it is clear that one of the main objectives of the Cloud computing paradigm is an efficient use of dormant resources in the infrastructure of these big companies. According to some [1], cloud computing is the next paradigm shift in Information Technology, and it is also said to be the long-held dream of computing as a utility, with the potential to transform a large part of the IT industry [2].

The Cloud concept offers advantages to both the Cloud providers and the Cloud user. On one hand, the provider needs to focus on assembling infrastructures providing high-performance computing and high-capacity storage to offer as utility computing in a pay-per-use basis. On the other hand, the customer is able to take advantage of state of the art services and infrastructure with no extra expertise requirements, which assures him fewer expenses with personnel and, at the same time, removes the problem of over provisioning, i.e., investing in computing resources that it does not need or that it only needs infrequently.

Another comparison that is usually made is between Cloud Computing and foundries in the hardware industry [2]. Semiconductor foundries are companies that build chips for third parties, e.g., Taiwan Semiconductor Manufacturing Company (TSMC). Foundries enabled small creative companies, with no financial capacity to own a private fabrication line, to implement their ideas and thrive in the hardware business. We can now be at brick of this shift but at the services and software level.

There are three new aspects in Cloud computing that are the core of its advantages from a hardware point of view. The first one is the illusion of unlimited computing resources available on demand for the cloud user. This property allows cloud users to put their provisioning concerns in secondary place in the list of priorities. Second, is that there is no need for an up-front commitment in terms of resource acquisition from the cloud user. This capacity derives from the first property because if hardware capacity is not an issue, a company can augment or reduce its needs according to its current growth state. The third, and last aspect, is the ability to provide a pay-per-use fashion for computing resources. Again, all these three properties are interconnected with each other, which tell us that we can use computing resources in an as needed basis instead of making big investments in huge infrastructures that might end up having a very low percentage of use.

The concept of Utility computing is defined as the service being sold in a public cloud infrastructure, which is seen as a pay-as-you-go cloud solution to the general public [2]. Utility computing is going to be able to provide enough capacity for a very popular novel Internet service, like it is also going to be able to minimize costs if the service is not as successful as it was thought to be. Companies can see cloud computing as the opportunity of having

a cost-efficient test bed for their products.

NIST has defined four different deployment models for the Cloud [3]. In the document, we can read, *"This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.".* We have public, private, community and hybrid Clouds. We are just going to provide a short description of each one of the models so the differences between them become clear. The essential characteristics of the Cloud have already been discussed in the previous paragraphs (e.g., *measured service*) and the three service models each one has a brief subsection after this introductory text.

The *public cloud* concept encompasses Clouds that are maintained and run by companies specialized in offering Cloud computing solutions. They are considered public because they can have running tasks from different customers all executing in the same physical infrastructure. We can also have *private clouds*. This second type is different from the first in that it is owned by the cloud user, which is also the only user of the infrastructure. This solution is more expensive because the cloud user has to make a considerable investment in infrastructures and also deal with the costs associated with its maintenance. It is true that the advantages of a public cloud infrastructure are lost in this scenario, but it can be theoretically considered more secure because it is not shared and it is under the control of its only user, the owner. This type resembles current privately owned infrastructures. The third concept defined in the document, *community cloud*, is similar to a private cloud but instead of having a single user it has a group of users that have shared concerns (e.g., mission, policy, and compliance considerations). Finally, we have the *hybrid cloud* notion. This is where we combine the first two concepts (i.e., public and private clouds) and obtain a cloud where some parts are shared and others are owned by a particular user. This type will be the one posing more challenges in terms of maintenance.

Although the Cloud paradigm is being embraced by a considerable part of the IT industry (e.g., IT giants like Microsoft, Google and Amazon are examples of companies investing in Cloud solutions at various levels of abstraction as we are going to discuss later), some people merely call it a marketing hype campaign. In the near future, the impact caused by Cloud computing in the IT industry will tell us which end is right, although the move of many companies to the cloud seems to be irreversible.

Subsequently, we define the three different service models considered for Cloud computing. We are referring to the concepts, starting from a lower to higher layer, of *infrastructure as a service*, *platform as a service* and *software as a service*. We are presenting these definitions in accordance with [1] and [3]. In [1], Sun presents an interesting view of Cloud computing, in the text it is considered as the agent of change in the adoption of a new and lighter web stack. In its proposal, Sun mentions solutions such as the open-source

web server lighttpd, an open source framework for data-intensive distributed applications known as Hadoop [4], and MogileFS [5] a powerful distributed file system.
Sun also discusses how this new paradigm can be seen as a new architectural layout for the Internet, offering a novel application development environment composed by a new set of layers, infrastructure, services and applications.

2.1.1 Infrastructure as a Service

In the Sun Microsystems view of the Cloud, this is the lowest layer in the Cloud infrastructure architecture. At this level in the architecture, we have the fundamental components that give the infrastructure its storage and computing capabilities, which will then be offered as a service to the cloud user. This can be seen as the layer providing the physical resources to be sold in a pay-per-use manner, also known as Utility Computing.
A well-known commercial product that offers solutions at this level is the *Amazon Elastic Compute Cloud* or *Amazon EC2* [6]. This solution provides the customer full access and control over the computing resources he paid for. This does not mean that the cloud user has control over the underlying cloud fabric, but that he has control over a virtual machine, or a set of resources, running on top of the Cloud fabric controlled by the Cloud provider. In this setting the cloud user is then free to configure their virtual machines with whichever solutions he sees fit. This can be seen as the layer where the level of freedom of the user is the highest. At this layer the Cloud user still has to be concerned about maintaining the software he chooses to install in the resources rented to the Cloud provider.

2.1.2 Platform as a Service

The second level of abstraction in Cloud computing is known as *Platform as a Service* or *PaaS*. In this level, we tend to have an already configured group of services that when combined offer a development environment to be used by a Cloud developer to generate *SaaS* solutions.
A classic example of *PaaS* is a virtual machine image containing a set of software services (for example, a Linux distribution, a Web server, and a programming environment such as PHP) in order to offer a web development environment for the Cloud developer. This configuration is similar to the solutions offered by Web hosting companies. The difference might reside in the way the Web hosting company manages its infrastructure. If it is like a Cloud, then the Web hosting company can already be considered a Cloud provider, if it is not it is a simple Web hosting provider.

Some commercial examples from relevant companies in the IT field are already available. From Microsoft we have the Windows Azure Platform [7], while Google offers the Google App Engine [8]. The Cloud developer can use these platforms to simplify its implementation process by relying on the set of pre-defined tools offered by them. Although these platforms can provide a considerable amount of flexibility, the limitation at this level is that the developer is constrained by the functionalities offered through these platforms.

2.1.3 Software as a Service

Software as a Service or *SaaS* is the highest level of abstraction in the Cloud. A solution offered at this level is an application ready to be used by a Cloud user. These are usually on-demand and multitenancy applications. The multitenancy property comes from the fact that they are usually a single instance of the software, running on top of an infrastructure of the Cloud provider, simultaneously serving multiple client organizations or tenants.
A leader in the industry of *SaaS* is Salesforce.com [9], who is offering multitenant solutions in the field of *Customer Relationship Management (CRM)* since before the appearance of the concept of *SaaS* in the context of Cloud computing.
A more recent example of this type of solutions is the e-mail service offered by Google, i.e., GMail, through its Google App Engine [8]. In these situations the Cloud user is only interested in getting the most out of the application provided by the Cloud. At this level the Cloud user is not seen as a developer anymore, he is a simple user of solutions offered by Cloud developers.

2.2 Security in Cloud Computing

Cloud computing security is a research topic receiving a considerable amount of attention. Although security is a recent focus of the research community, we possess a few examples to illustrate some of the already encountered security issues. We are going to present a powerful external attack that can be mounted against the cloud infrastructure, and discuss on the impossibility of using cryptography as a one man army to solve all the security challenges inherent to the Cloud. Another point we are going to include in this subsection is a recent discussion regarding the viability of implementing an integrity-protected hypervisor. This is relevant because the Cloud infrastructure is going to need an integrity-protected hypervisor to assure some security properties to its users.
Our contribution deviates from the examples we present in this section. To the best of our knowledge there is few or no previous work discussing the issues we expose in this text.

Throughout this document, we will provide an analysis with respect to privacy issues, in current cloud infrastructures, from the perspective of a malicious insider. We will list our assumptions and describe how serious is the impact of the actions taken by a malicious insider regarding its ability to compromise privacy in current Cloud computing infrastructures.

2.2.1 External attacks against the Cloud infrastructure

The external attack we just mentioned is a powerful co-residency attack mounted by an external malicious entity as can be seen in [10]. This attack has a considerable impact on the security properties of the cloud infrastructure. The attack consists in mapping the internal cloud infrastructure to identify where target virtual machines might reside. This information is then used to instantiate malicious VMs until one of them ends up co-residing in the same physical machine with the target victim virtual machine. After this stage, the attackers have their malicious VMs running over the same lower layers as the victim virtual machine. The attackers can now use cross-VM side-channel attacks to monitor or extract confidential information from the target VM violating the privacy of data running in the victim VM. These attacks are serious threats against the security of the cloud infrastructure.

2.2.2 Applicability of Fully Homomorphic Encryption (FHE)

Another important aspect regarding the security of the cloud infrastructure, that needs to be kept in mind, is that cryptography on its own is not capable of solving all the security issues related to cloud computing. Recent work shows that even fully homomorphic encryption (FHE), also known as the holy grail of cryptography, is not able to solve all the security challenges posed by the cloud infrastructure [11].
Fully homomorphic encryption is a method that enables computation to take place over encrypted data. In a cloud environment, for example, such a solution, if the practical performance issues are solved, is going to allow a client to encrypt data and send it to the server for processing. The server then performs the required operations over the encrypted data, without decrypting it, and sends the results back to the client without ever learning the contents of the confidential data. The authors define three cloud-application classes, the private single-client computing, private multi-client computing and private stateful multi-client computing. FHE can in fact be used in private single-client applications where the user outsources computation to the cloud. The problem is that the strength of the cloud is in multitenancy applications, and in those scenarios we are in a multi-client architecture. So,

the single-client computing class is considered limited and we need to recur to multi-client classes if we want to obtain significant advantages from the cloud. This leaves us with the two multi-client classes the authors describe. Unfortunately, they also demonstrate that FHE cannot be used to assure security in those scenarios. They conclude that the FHE solution is limited and leave the security challenges inherent to multi-client classes as an open research problem.

2.2.3 Integrity-protected Hypervisor

The research community is working hard on the topic of integrity-protected hypervisors and related problems [12][13][14]. Having an integrity-protected hypervisor means that the hypervisor's code cannot be altered in any fashion and that the hypervisor's data cannot be maliciously changed. The objective of having such a tool is to offer code integrity as well as data integrity and secrecy when executing applications on top of the integrity-protected hypervisor. These authors intend to reach such a solution because it can be used as a security foundation for many different situations. In our case we are interested in how it can be applied to solve existent security challenges in the current cloud infrastructure.
The problem is that recent work reveals some negative conclusions [13]. In this article the authors propose a set of rules that should be followed by hypervisor developers and users so that the final outcome is a truly integrity-protected hypervisor. After describing the rules and how they need to be followed in order to avoid security threats they also present considerably negative conclusions. They conclude that in practice current x86 hardware is not capable of realising an integrity-protected hypervisor. This fact holds true albeit in theory latest x86 hardware contains enough technology to support an integrity-protected hypervisor. Another conclusion is that an integrity-protected hypervisor will not be able to support legacy guests. In this context, a legacy guest is a legacy operating system executing in a virtual machine. The problems raised by legacy operating systems are due to the complexity associated with implementing certain modes of operation (e.g., BIOS calls), which would increase the complexity of the hypervisor causing vulnerabilities in the supposedly integrity-protected hypervisor. They also recommend care when selecting hardware to assemble a system to protect hypervisor integrity because not all hardware devices are going to be compliant with the requirements to support such solutions. All these arguments tell us that there is still a lot to do at the lowest layers if we want to be able to achieve a high level of security in the cloud infrastructure.

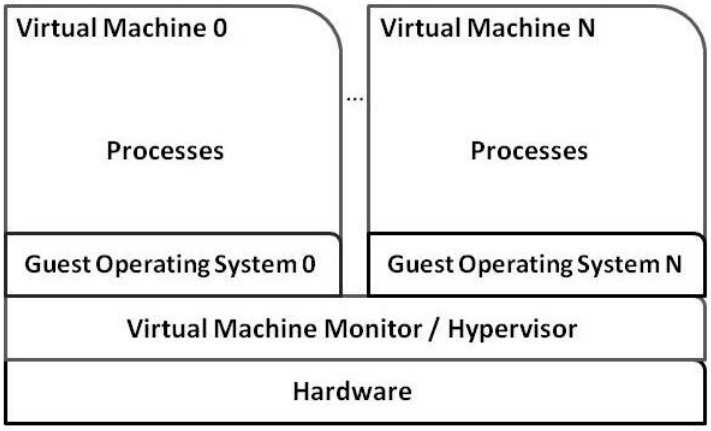

Figure 2.1: Native Virtualization

2.3 Virtualization

Although the concept of virtualization is currently a much discussed topic, it dates back to the late 60s when IBM gave birth to the concept in their *mainframes*, e.g., System/360. Virtualization is a concept that is closely related to the notion of abstraction [15], because it has to do with a layer of abstraction over a certain group of physical resources. This abstraction is going to allow multiple tasks or users to take advantage of those resources without noticing each other. A clear-cut example is the Java virtual machine concept. This JVM is the layer that provides a virtual machine to programs in *bytecodes* where they can execute. In this text, we are concern with the concept of a *virtual machine monitor (VMM)* or *hypervisor*. The *hypervisor* is a component that allows the abstraction of system resources in form of an emulated hardware interface, which can then be used by an operating system executing on top of it in a *virtual machine (VM)*. In this manner the hypervisor is capable of sharing the available resources with multiple virtual machines.

2.3.1 Virtualization types

When we identify a virtualized environment, we can be referring to native or traditional virtualization, or a hosted virtualization. These are currently the two existing types of virtualization. We provide an illustration of each virtualization environment in Figures 2.1 and 2.2.

In this first image we can see the architecture for native virtualization. In this type of virtualization we simply have a virtual machine monitor, over the hardware layer, controlling the virtual machines that are taking advantage of the resources available at the hardware level.

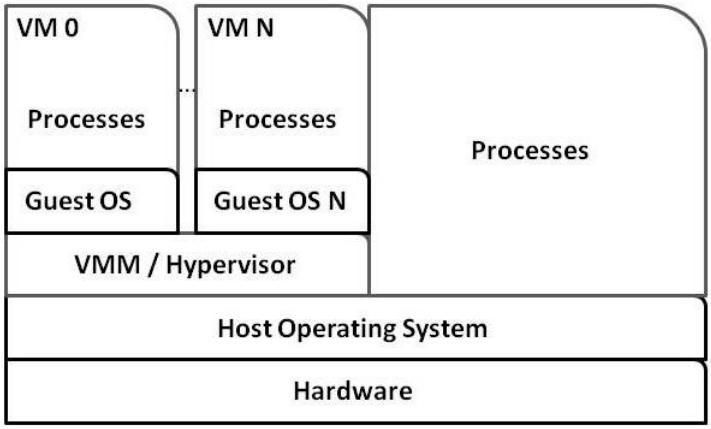

Figure 2.2: Hosted Virtualization

This type of virtualization brings an advantage in terms of lower performance overhead, because the only layer between the operating systems operating in the virtual machines is the virtual machine monitor. Native virtualization is the type of virtualization that is used in Cloud computing because it allows multiple virtual machines to share a server with high efficiency. A popular virtual machine monitor used in this type of virtualization is Xen [16], which we will discuss later in this section because we will be using it in our test environment.
In the picture below we have the second type of virtualization. In the hosted virtualization architecture the virtual machine monitor runs within a hosting operating system that is executing directly on top of the hardware level. In this case the user has its running operating system and then can use the virtual machine monitor to configure multiple virtual machines using different operating systems. A recognised player in this type of virtualization is the VMWare workstation solution commercialised by VMWare [17].

2.3.2 Security applications

We are now going to discuss some ways in which we can use virtualization as a security mechanism. In order to be useful security wise, the virtual machine monitor needs to assure some properties [15].

Completeness A hypervisor needs to assure that a virtual machine is not capable of accessing system resources or a different virtual machine without going through the hypervisor.

Isolation This property offers guarantees that we have a mechanism to execute different applications isolated from each other. Let us imagine that an application crashes in

a virtual machine. To assure isolation the hypervisor cannot be affected by this crash and in this manner leave applications running in different virtual machines unaffected. We can easily see that this property as strong security applications. For example, if one application is compromised it cannot affect other applications taking advantage of the virtual environment.

Transparency The transparency property is the ability of keeping the software running inside a virtual machine from learning that is executing in a virtualization environment. The software must not be able to detect that is being executed by an operating system that runs over a hypervisor, and not in a conventional configuration where it usually executes over an operating system that is operating directly over the hardware layer.

Combining these properties makes virtualization a useful tool in some specific scenarios for a security researcher or professional as we are going to discuss with a couple of examples.

2.3.2.1 Isolation and Recovery

This mechanism is applicable to threat scenarios that can occur within a single machine or a distributed system. The role of virtualization in these scenarios is to provide modularity through the form of virtual machines. This modularity can then be used to isolate compromised modules of the system and launch a recovery process. For example, let us imagine that we have three distinct applications running each in a different virtual machine and an attacker is able to compromise one of these applications. This event can be controlled by freezing the compromised virtual machine and relocating it to a different physical machine where it can be analysed in order to reveal valuable information concerning the vulnerability or vulnerabilities that allowed the attacker to compromise the application in question.

2.3.2.2 Honeypots and Introspection

A *honeypot* can be seen as the sophisticated and digital equivalent of the rudimentary physical mouse trap. In this digital era we are preparing a trap for malicious computer users or computer malware. The *honeypot* is usually composed by a set of machines that pretend to appear a normal production computer system to the outsider. What the outsider does not know is that these machines are isolated virtual machines that are left with some known vulnerabilities, the *digital cheese* or *honey*, on purpose. When we have a *honeypot* composed by several physical machines, it is also sometimes referred to as a *honeynet*.

Since the purpose of setting up honeypots is to study the behaviour of malicious computer users or automated malware, we then need a mechanism to extract useful information from it. This is where *introspection* comes in handy. Introspection techniques are able to observe events that took place within a virtual machine, which is ideal to monitor and learn from the behaviour of our study targets. There are multiple introspection techniques currently available. We are just going to briefly describe them, if the reader is interested in obtaining more details with respect to these topics please consult [15] and [18].

Monitoring agent inside the virtual machine In this type of introspection there is a software module operating inside the virtual machine that we wish to monitor, responsible for obtaining the desired information. The software module is usually integrated as a module of the operating system kernel operating in the virtual machine in question.

Breakpoints and inspection This technique is less intrusive than the previous one because it does not impose any changes to the operating system running on the virtual machine. The idea here is to add some breakpoints in the VM code so it can capture the desired events, similar to the usual debugging. This has clear limitations because it is dependent on specific configurations.

Checkpoint and rollback This third introspection mechanism is divided in two stages. The checkpoint phase consists of capturing the current state of a virtual machine. The rollback step is used to return to a state previously obtained using the checkpoint functionality. The *checkpoint* and *rollback* capabilities are supported by some virtual machine monitor[15]. The mechanism starts by obtaining a checkpoint of the target virtual machine, introduces some analyses code, and when it is done performs a rollback operation to the previous state.

Architectural This last technique is not as intrusive but it is also very limited. The gist of it is to develop a personalised hypervisor in order to read the events that come from the virtual machines that we wish to monitor. This solution is limited to the events that go through the VMM.

Virtualization is a foundation of Cloud computing and so we dedicated the time to present it with enough detail to the reader. It is also important to mention that we see it only as one of the components of the Cloud. We are not interested in studying virtualization security issues in detail. We have learnt about virtualization types and some usual security applications of virtualization. Hopefully, this information will make the Cloud computing section text clearer. With the same intentions in mind we conclude this section on virtualization with a presentation of the Xen hypervisor which was the hypervisor used in our test environment.

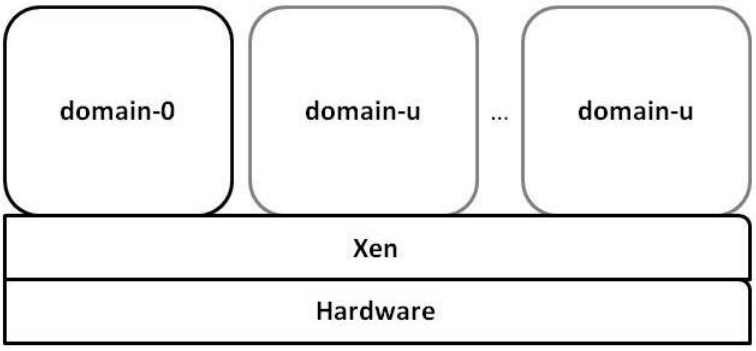

Figure 2.3: Xen Architecture

2.3.3 Xen

The Xen hypervisor is a thin layer of software that operates directly over the hardware of a machine. Its purpose is to replace the conventional operating system so the resources of the machine can be efficiently shared among multiple virtual machines. This is an example of the native virtualization type we described early, which means that we can have multiple operating systems running in virtual machines that execute over the Xen hypervisor.
The Xen hypervisor has two types of virtual machines operating on top of it. One has a unique instance and is known as the *privileged domain* or *domain-0*, the other type is the *unprivileged domain* or *domain-u* and can have multiple instances. The later type cannot be launched unless there is already a privileged domain running. Figure 2.3 illustrates this architecture. We now describe these types of domains and their specific properties.

Privileged domain: the privileged domain or domain-0, as its name suggests is the privileged control domain which means that it has privileged access to the Xen hypervisor. These elevated privileges are reserved to the domain-0 and no other domain has the same access level to the system. An administrator with access to the privileged domain can control the whole system, e.g., launch or remove unprivileged domains. The privileged domain is launched by the Xen hypervisor at startup and can run any operating system except Microsoft Windows [19].

Unprivileged domain: an unprivileged domain or domain-u is a guest that is launched by the privileged domain and executes independently on the system. Currently, there are two different types of unprivileged domains. We can have domain-us that run under *paravirtualization* or *hardware virtual machine* or *HVM*. The requirements to have an unprivileged domain that runs Microsoft Windows are to be able to run HVM domain-us.

Paravirtualization: This concept is associated with an operating system that has been modified in order to be able to operate over a hypervisor instead of directly over the hardware layer.

Hardware Virtual Machine (HVM): In this case, we have an unaltered operating system that is executing in a virtualized environment unaware that it is not running directly over the hardware layer. In these scenarios we require special hardware (e.g., the already mentioned Intel-VT and AMD-V).

2.4 Trusted Computing

Trusted computing is an emergent security technology trend related to the work of the Trusted Computing Group (TCG) [20]. The need for this type of technology advents from the increasing security issues in the information technology industry. As a computer user we usually tend to trust the software controlling our machine, for example, the operating system, if it is a genuine Microsoft Windows copy the user is satisfied by that indication. A similar scenario might occur if a user downloads a Linux distribution from the Internet and installs it on his machine. The problem starts when this software might not really be doing what it states to be. In these situations, where do we obtain a root of trust to assure that the software operating in our machine is really what it claims to be? This is where trusted computing comes in handy with its *Trusted Platform Module* providing roots of trust.
We use the following subsections to describe the architectural elements we consider more relevant in the TPM specification. We assume that the information we provide in this section provides enough background to follow the remainder content of this document. For a complete introduction to trusted computing and related components we refer the reader to [21] and [22].

2.4.1 Trusted Platform Module

The *Trusted Platform Module (TPM)*, as defined by the TCG, is a microcontroller responsible for storing keys, passwords and digital certificates [23]. The advantage of using the TPM to store this information is that it is safer against software attacks and physical theft. The TPM is used as *root of trust* in trusted computing. A *root of trust* is a hardware or software mechanism that is implicitly trusted by the computer user [22]. We can identify three roots of trust:

Root of trust for measurement (RTM): a trusted implementation of a hash algorithm (e.g., SHA-1) that, depending on the platform, may or not reside within the TPM, used to

provide accurate system measurements that will subsequently be used to put the system in a trusted state.

Root of trust for storage (RTS): the TPM offers a *protected* storage area for one or more secret keys – most of the times it is a single key, the *storage root key* or *SRK*.

Root of trust for reporting (RTR): the root of trust for reporting a single key stored in the *protected* storage area within the TPM. This key is called the *endorsement key* or EK and uniquely identifies the corresponding TPM in which it is securely stored.

These components are considered the roots for trust because they are indivisible [21]. They are the base of the *chains of trust* that are generated in order to establish a trustworthy system. It is obvious that these roots of trust have a key role in the concept of trusted computing.
Both, SRK and EK use asymmetric cryptography. The TPM is supposed to securely hold the secret part of these two key pairs. Usually, the public key of the EK is signed into a certificate by the TPM manufacturer at development time. Under normal circumstances the EK key pair is maintained throughout the lifetime of the hardware TPM module. The SRK is a little different. Albeit it is also safely kept by the TPM, it is only created when someone takes ownership of the TPM and altered if ownership of the TPM changes.
Another important concept is the *Trusted Computing Base* or *TCB*. This concept was introduced by the US Department of Defense (DoD) in their *1985 Orange Book* [56]. The TCB is defined as being the totality of protection measures within the system, those parts of the system that we rely on, and whose failure almost inevitably leads to a compromise state.

2.4.2 Platform Configuration Register (PCR)

The PCRs are important players in the concept of trusted computing. They are registers responsible for holding the measurements of the system that are reported to the TPM. These measurements are then used in multiple secure operations provided by the TPM. Even though we can directly read a PCR, in order to assure the safety of performed measurements, the PCRs can never be written directly to. The solution is the use of the *extend* operation. This operation consists in performing a cryptographic hash of the measurement already resident in the PCR combined with the new measured value submitted to the TPM. A set of *extend* operations generates a particular state in the affected PCR, this state can later be used to conclude about the integrity of the platform in a process denominated *attestation*. The process of *attestation* consists in providing the system challenger with a signed measurement of the current state in a particular TPM.

2.4.3 Attestation Identity Key (AIK) pair

As we have already mentioned the *root of trust for reporting* or *endorsement key* is a relevant component of the TPM, and under normal circumstances it will be part of the TPM for its useful lifetime. This fact raises a considerable privacy issue when using the EK to sign PCR measurements used to attest the platform in question because every attestation can be linked backed to the particular EK that provided that signed information.
To solve this problem the TPM specification introduces the ability to generate *attestation identity key (AIK)* pairs, that when created, are signed by a Privacy CA in order to build a chain of trust. They can also be signed through a *direct anonymous attestation* solution (more on this topic later). The TPM can generate as much of these keys as necessary. This process is going to severe the ability to link attestations back to its originating TPM. This is very important to keep the privacy of a particular TPM.

2.4.4 TPM functionality

In this subsection we intend to give the reader a couple of examples to illustrated how we can advantage of the functionalities offered by the TPM.
An interesting functionality is *sealed storage*. This operation consists of a method that by combining the RTS, and the measurement values stored in PCRs, is capable of protecting external data using encryption. There is a *sealing* and an *unsealing* operation. The *sealing* operation is the process of protecting the external data. It uses the external data, a requested PCR value and encrypts the data. The final result is a sealed data package that can only be unsealed by the same TPM that created it. The TPM ensures this property by including a nonce, known only to it, in the sealed package [22]. The *unsealing* operation is not quite the opposite of the sealing operation. This is true because any TPM can perform a sealing operation while the unsealing process is restricted to the creator of the sealed package. The requirements are that the TPM trying to unseal the data must possess the right nonce and the correct PCR measurement in the used PCR. If these conditions are not verified the unsealing operation is aborted and no access is granted to the protected data.
Our second example is the ever famous *attestation*. The use of attestation comes in handy when an external entity trying to verify the state of a platform has previously not performed a sealed operation. Recall that the unsealing operation would assure the external entity that the current state of the remote machines is the state present at moment the sealing operation took place. When in this situation the remote party recurs to attestation to verify the current state of the platform before establishing trust with it. Attestation consists

in providing a digital signature of one or more PCRs and having the remote party validate the signature and the PCR values. After validating this information the remote party establishes trust with the platform, known to be in a secure state.

2.4.5 Direct Anonymous Attestation (DAA)

The concept was born from jointly work performed by IBM, Intel, HP and the TCG. The outcome of that work was published in [24]. In the referred document we can read that the TCG adopted DAA as the method for remote authentication of the TPM hardware module, while preserving the privacy of the platform user. This concept was created to deal with availability issues existent in the *privacy certification authority* or Privacy CA approach.
The Privacy CA approach consists in assuming that the Privacy CA knows the public keys of the EKs of all valid TPMs. Using this information the Privacy CA can then be used as an authenticator for AIKs generated by valid TPMs. If the Privacy CA verifies that the public portion of the AIK is valid with respect to its EKs list, it issues a certificate for that particular AIK pair. The TPM can then use this certificate to authenticate its AIK pair.
This description confirms the heavy load that would be put on the Privacy CA side. This would definitely raise availability issues as concluded by many privacy groups and data protection commissioners [25].
DAA can be seen as a group signature scheme without the tracing feature [24]. A group signature scheme allows a group member to sign messages anonymously on behalf of the group [55]. Group signature schemes have a tracing feature, i.e., the identity of a signer can be revealed in case of dispute. The DAA mechanism uses a signature scheme to issue certificates on a membership public key generated by a TPM. To authenticate as a group member, a TPM needs to prove that it possesses a certificate on a public key to which it also knows the private key. For the verifier to be able to detect rogue TPMs, the TPM is further required to reveal and prove correct of a value *Nv = zeta^f*, where *f* is its secret key and *zeta* is a generator of an algebraic group where computing discrete logarithms is infeasible. The authors solved two important problems concerning the issuers of certificates. The problems related to using safe primes when obtaining the RSA modulus and, chose a rather large modulus to make the discrete logarithm problem hard for the issuer as well.

2.4.6 Trusted Computing Group Software Stack

The *trusted software stack* or *TSS*, as the TCG puts it, is a software specification that provides a standard application programming interface for taking advantage of the functions

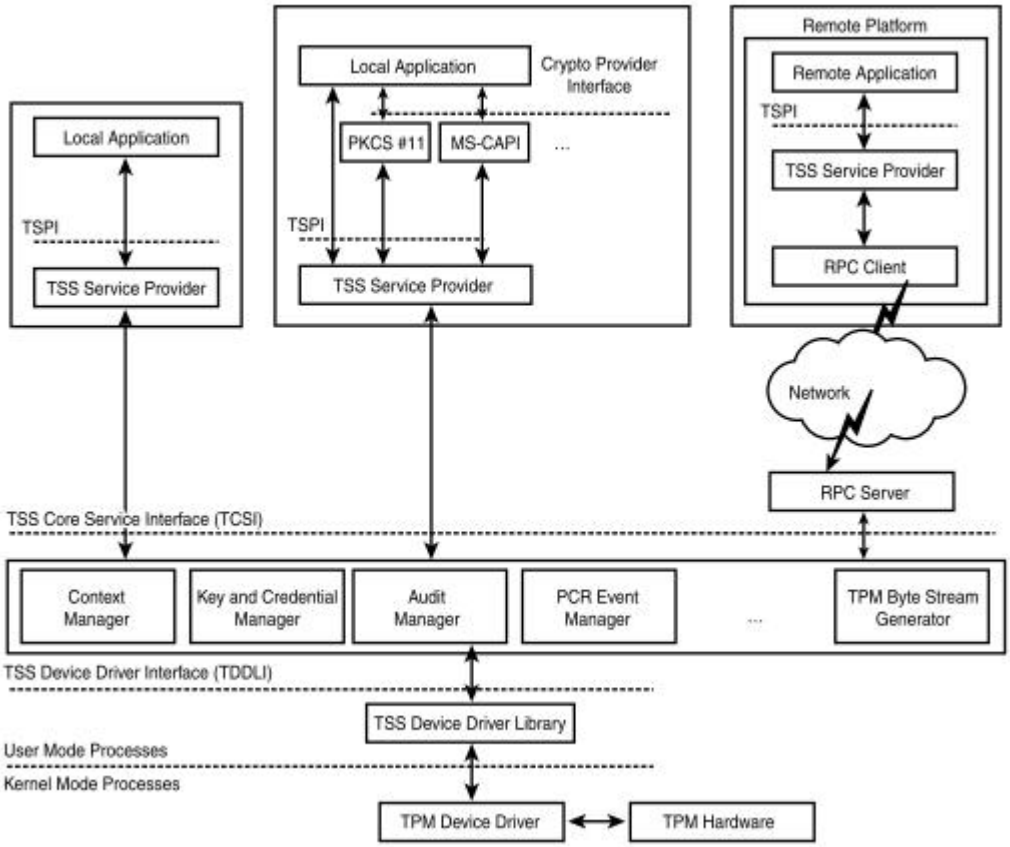

Figure 2.4: Architectural overview of the TSS

offered by the trusted platform module. The main objective is to simplify the access to TPM functionality and have a software specification that allow developers to create interoperable client applications that are vendor-independent. In our work we used an open-source TCG software stack denominated *TrouSerS [50].*

The three logical components of the TSS are the *TCG device driver library* (TDDL), the *TCG core service* (TCS), and the *TCG service provider* (TSP). We provide an architectural overview of the TSS in Figure 2.4, the figure is courtesy of [49].

The TDDL is a library thar offers an application programming interface to interact directly with the TPM device driver. For example, it provides a set of application programming interfaces to open and close the device driver, send and receive data blobs, among others.

On top of it we have the TCS layer that has several functions. It manages the TPM's resources, such as authorization session and key context swapping. It is also responsible for the TPM command blob generator, which converts TCS API requests into the byte streams the TPM is capable of understanding. Synchronizing application access from the TSP layer and system-wide key storage are also part of its capabilities.

The highest layer of the three logical TSS components is the TSP layer, this layer interacts directly with the application and is implemented as a shared object or dynamic link library. The TSP interface or TSPi exposes the TPM's capabilities and some of its own, e.g., key storage and pop-up dialog boxes as input for authorization data.

Chapter 3

Analysis of the Privacy of Cloud Computing

In this chapter we demonstrate that current cloud computing infrastructure solutions are not capable of enforcing the privacy of secret data entrusted to the cloud provider by the cloud user. Cloud providers are faced with the challenge of proving to their clients that their cloud infrastructure as a whole can be trusted. This is one of the most fundamental and pervasive security challenges of cloud computing because when the user agrees to use the cloud infrastructure, he relinquishes control over the physical and virtual infrastructure used to manipulate its private data. This creates an array of possible attack vectors that can be exploited by a malicious entity, internal or external to the cloud infrastructure. We are going to focus in internal threats as we will state in the attack models we present along the chapter. We also provide proof showing how some security measures used to restrict the tool set to which the attacker has access are not effective against a powerful and determined attacker and, even more important, how some of those restrictions might not even be possible to implement using current technology.

In order to perform the security experiments necessary to obtain the demonstrations for the attacks we describe in this section, we had to mount a test setting that reproduced, in a very small scale, only one or two computers, a cloud computing infrastructure environment similar to the ones we can find in industry used to offer Infrastructure as a service to cloud users. We have used the latest versions of the Ubuntu Server distribution together with the Xen hypervisor to assemble such an environment, details regarding on how to reproduce our experimental environment can be found on Appendix A.

Basic Attack Model

We are assuming the most powerful adversary that cloud computing infrastructure security

can face, the security-educated malicious system administrator. Our malicious administrator is going to have the capacity of locally, and remotely, logging in into every physical machine within the network with root privileges. He is also going to be able to gain physical access to the machines when deemed necessary. With these levels of access he is going to have a myriad of attack techniques and tools at his disposal to compromise the security of the cloud infrastructure. It is also assumed that the malicious insider has no access to the virtual machines operating on a particular physical machine, we consider that the virtual machines belong to the clients.

If no indication on contrary is provided at the beginning of the subsection where an attack description is presented, it is assumed that the attacker has complete control over an hypervisor through the privileged domain. This means that the hypervisor will have a set of tools that could aid the attacker in achieving his final objective of violating the privacy of unprivileged domains of the cloud infrastructure.

Since the administrator has root and physical access to the machines, we are assuming that he is capable of finding ways around logging systems that are running on the machines.

We are not considering attacks mounted from the outside, of the cloud infrastructure perimeter, against the targeted infrastructure. Our focus in this work are attacks against the confidentiality/privacy of the cloud infrastructure.

Scenario

The server physical machine in which we are going to perform the attack is part of the cloud computing infrastructure of company INI-Cloud. The company offers cloud computing solutions. The machine was already booted up using the latest version (at the time of our experiments) of the Xen hypervisor, 4.0.1-rc4, as its privileged domain or domain-0, and it is also using a Xen enabled kernel with version 2.6.32.15. The model of the physical machine used is a Toshiba R10-120. The Xen hypervisor boots and launches the privileged domain. No unprivileged domain may run unless the privileged domain has already been booted up. The domain-0 is a modified Linux kernel as we have mentioned in Section 2.3.3. The Linux boot sequence remained unaltered for both domain-0 and unprivileged domains. This means that the information and software usually loaded during the boot process is similar in a privileged or unprivileged domain, when compared to a Linux system booting in a usual machine that does not have a hypervisor layer between the operating system and the hardware. The Gnome desktop software was installed in the domain-0 but not in the unprivileged domain. The domain-0 was used by one of INI-Cloud's administrators to setup an unprivileged domain or domain-u for a specific client that we discuss below.

The client, Cloud-Bank, requested this virtual machine to configure a server in order to host a home banking solution it wishes to make available for its clients. The client installed a Linux Ubuntu Server 10.04 distribution, with Apache web server version 2. The Apache web server was not configured to be launched at system start up. We launched the web server when we deemed necessary. The web server was configured to establish secure connections using the SSL communication protocol. A private-public key pair was created to allow the secure communication feature (the reader can find a detailed explanation of the steps to configure it in Section A.5). The loading of the private key was protected using a passphrase, which means that the key file, stored in disk, had its contents encrypted. We did not program any type of special index web page for the Cloud-Bank home banking solution. The action of establishing a secure connection with the unprivileged domain's web server, was assumed to be a connection where an initial page for the home banking solution would be loaded. The Cloud-Bank administration board was not aware of the security issues they ignored when they made a decision that only took into account the economic advantages of resorting to cloud computing as an infrastructure to host their home banking solution. A series of events would soon enough make them realize how serious the mistake they had made was.

With the process of setting up the cloud infrastructure environment concluded, INI-Cloud realized it should let go one of their top IT administrators because his salary was considerably high and in a time of economic crisis they needed to cut on personnel expenses. They believed the hardest load of the work was done and that their junior IT administrators would be capable of maintaining the cloud infrastructure operational from now on. So it was, they informed Joe, the IT administrator in question, with a few months notice of their intentions of letting him go.

After receiving INI-Cloud's contract rescission notification, Joe was not very happy with the situation because he was one of the first IT administrators to join the team and, in his opinion, a key player in the development of the company. INI-Cloud never expected Joe would have the reactions we subsequently describe.

3.1 Clear text passwords in Linux memory dump

This first attack and the next one are closely related because they both take advantage from the fact that we can easily obtain a memory dump from an unprivileged domain. We only need to know how to use the set of subcommands available for the Xen management user interface (a subcommand is the first argument passed to the command that invokes the Xen management user interface). When we have this knowledge it is fairly easy to obtain

a live memory dump from an unprivileged domain. To do so, the administrator simply needs to have root access to the privileged domain responsible for interacting with the hypervisor. Obtaining a memory dump from unprivileged domains can be seen as a regular procedure included in a forensics investigation, where usually the administrators are trying to figure out the cause for an unexpected failure on a specific unprivileged domain. So, in current cloud computing providers this type of operations is looked upon as legitimate by their administrators. This level of access to the system, and its infrastructure, is a factor that greatly influences the ability a disgruntled administrator has to perform malicious actions against a cloud infrastructure.

The second part of the attack we present in this subsection has already been demonstrated in a different scenario in [26]. In his work, the author demonstrates that it is possible to extract passwords in cleartext from memory dumps obtained from machines running the Linux operating system. To some specific applications, e.g., TrueCrypt [27], the attack consists in finding byte patterns that come before and/or after the cleartext password so, in these particular cases, the search for passwords can be automated. The author leaves the automation problem for other passwords as an open research issue, and since we are not interested in demonstrating the complete attack, or finding novel techniques for automated password recovery from memory dumps, we are only introducing the state-of-the art on the subject. In our work, we are only focused in finding security flaws that a malicious cloud infrastructure administrator can exploit to subvert the privacy of cloud computing users. The attack is particularly interesting to demonstrate in a Cloud computing environment as we subsequently explain.

In our case, the attack simply illustrates that in a Cloud computing environment it is fairly easy to obtain the login password and the passphrase protecting a private key. A video of the attack can found in [58]. In this environment the private information discussed can also be retrieved from a memory dump obtained from an unprivileged domain running a Linux operating system. We are referring to the the login and root passwords and a passphrase used to decrypt the private key file, usually used by a web server, but this attack could also target other passwords used in the victim system. We are only mentioning this confidential data because in our environment it was everything we considered relevant to compromise in the scope of this particular attack. For example, if we had installed a database management system, compromising its passwords could also be useful in order to retrieve private information contained in a database.

This attack can be useful if the malicious administrator is interested in compromising the login password or in acquiring the passphrase used to, for example, decrypt a private key used by a web server. This second example of captured information can be useful when combined with the attack presented in Section 3.3. This combination would allow

the attacker to use the private key file, retrieved from the information captured in the attack described in the section we just referenced, even if it was encrypted.
Another way to exploit the strings found in a memory dump would be to perform dictionary attacks. It would take longer than directly retrieving the passwords or passphrases but it would also be considerably effective. This would only be considered a useful solution until novel automated techniques for retrieving clear text password comprising from memory dumps are discovered.

The Attack

Joe had a detailed knowledge of how the cloud environment works and that gave him an advantageous position to perform malicious activities. The scenario was as follows. In order to perform this attack we need to know some specific details on how the cloud environment operates and what constitutes it. In this situation, the required knowledge was to know that an unprivileged domain is executed on top of the same physical machine, or hardware layer, as its privileged domain. Some basic understanding of how a computer system operates is also necessary. Particularly, the attacker needs to know that programs performing delicate system operations, such as login password verification, reside in the volatile memory of the machine from where they are fetch to be executed to perform the task they were programmed to do. This operation method, combined with the low level of security surrounding the manipulation of passwords in memory, are the key points that allow an attacker to extract cleartext passwords from a memory dump.
Knowing these details, to perform the attack, Joe simply needed a method to obtain a memory dump from the unprivileged domain maintained by Cloud-Bank. There are several paths an attacker can follow that lead him to obtaining a memory dump from a system. We can use some tools that already come as integral part of a Linux distribution, such as *dd*, or other tools like *pcat* from the *The Coroner's Toolkit* [38]. From the various alternatives presented in [29], it is clear to understand that obtaining a memory dump can be very easy and stealthy with current technology. The authors of the cold boot attack describe the usage of small programs to effectively produce memory dumps that can then, for example, be stored for transport in USB drives or iPods.
We have been discussing the memory dump issue in usual computer systems and architectures. In the Cloud computing infrastructure, to make matters worse, we have an easy to use subcommand denominated *dump-core* that is part of the Xen management user interface, *xm*, which we can use to obtain a memory dump from an unprivileged domain we desire to inspect or attack. The *dump-core* subcommand performs a dump of the memory region reserved to the targeted unprivileged domain. In Figure 3.1 we can see the command line where an attacker is using the Xen management user interface to obtain a

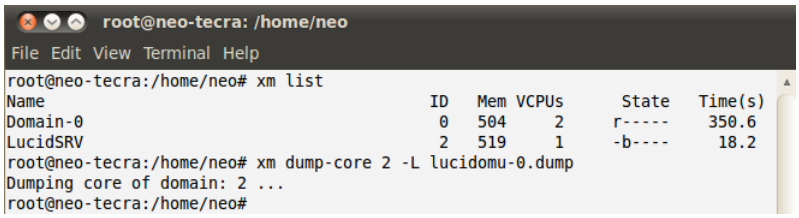

Figure 3.1: Live memory dump using Xen management user interface

live memory dump from an unprivileged domain running over the hypervisor controlled by the privileged domain to each the malicious administrator has root access. The memory dump contains all the information existent in that specific location of volatile memory, at the moment we request the privileged domain to execute the command. Since the privileged domain operates on the same physical as the domain-u, in the Cloud environment, as we will shortly explain, we do not even need to worry about the memory decay issue discussed by the authors of the famous cold boot attack.

The *dump-core* functionality of Xen is considered very useful because administrators can configure unprivileged domains to automatically perform a core dump when they experience crash failures. The information contained in these dumps is important in determining the error that caused the system failure. The problem is that we can also obtain live memory dumps from running unprivileged domains without affecting their operational status. The memory dump file generated by the *dump-core* command has recently been changed to be based on the ELF format because this format is considered easily extensible. We can find a description of the dump file and the sections it contains in the official Xen documentation [39]. For example, the file contains the contents of pages captured from memory in section *.xen_pages.* The corresponding frame number of a specific page is described in *.xen_p2m* section for x86 paravirtualized domains, and in *.xen_pfn* for full virtualized x86 and ia64 domains. For more details on the file the reader can consult the referenced document.

In Joe's case, he stored the memory dump of Cloud-Bank unprivileged domain in the *lucidsrv-2.dump* file as we can see in Figure 3.1. At this stage, if the attack performed in [26] is reproducible in the Cloud environment, Joe should already have in his possession passwords in cleartext that were resident in memory at the time he performed the memory dump. After demonstrating that in fact that information is in the possession of the attacker, the attack is complete.

We can see in Figure 3.2, that simply using the Linux *strings* command to extract the strings existent in the memory dump file, gives us the login and root password, we used the same password for both, and the passphrase used to protect the file holding the private key used by the Apache web server installation can be both found in cleartext.

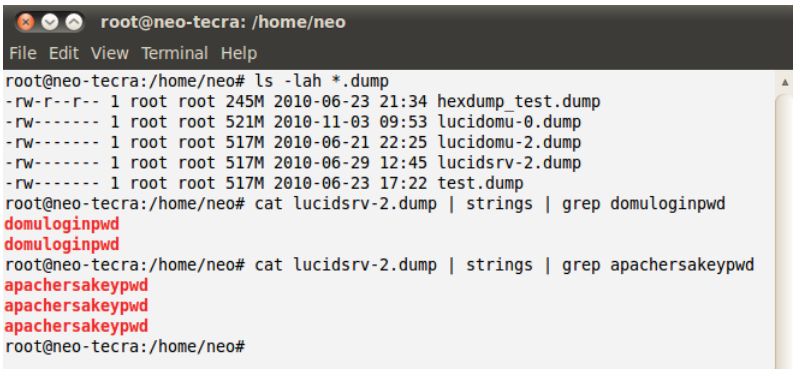

Figure 3.2: Clear text passwords found in memory dump

We could have also shown this the way it was shown in [26], performing an hex dump analysis on the dump file. In Figure 3.2, the first two occurrences outputed by the *strings* command are of the login and root password, which is *domuloginpwd*. It is displayed twice because we use it to login and also to obtain root access privileges to the machine, so its second appearance is when it is verified by the *su* command. The passphrase *apachersakeypwd* is only requested when the Apache web server is launching but it appears three times, this repetition must be directly related to the way the web server performs the verification of the passphrase. The outcome is that Joe was able to comprise confidential passwords that belong to the Cloud-Bank organization.

We did not go into much depth in reproducing the whole set of attacks described in [26] because we did not consider it necessary. The article provides a very clear explanation on how to obtain multiple passwords from a memory dump. We provide a figure that simply illustrates that, in fact, the passwords used in our test environment could be found in cleartext in a memory dump from that same system. Figure 3.2 presents the passphrase of the private key of the Apache web server and the login and root passwords of the unprivileged domain from which we obtained the memory dump. This proves that the attack [26] performed against a Linux operating system running on a non-virtualized environment is applicable to the cloud environment, which is the relevant aspect as already mentioned. We did not use the memory sniffing tool discussed in the original attack because we were not targeting passwords kept by TrueCrypt [27].

Pondering on the information contained in [26], and the proof we have presented in this subsection, we can conclude that the privacy of passwords belonging to the cloud user is not guaranteed by the current Xen cloud solutions. This fact can have really bad consequences for the cloud user and provider has we illustrate in our fictitious attack scenario, where a disgruntled administrator, Joe, captures passwords from an unprivileged domain running a home banking solution

3.2 Obtaining private keys using memory snapshots

Specific Attack Model

The difference between this scenario and the one in our first attack is that here we are considering that the administration team responsible for configuring unprivileged domains knew about the cleartext passwords attack and took measures to protect the passwords resident in memory. They can have either stopped using passwords at all or used obfuscation techniques. The relevant aspect is that the attacker is considered to have lost access to cleartext passwords through a memory dump.

In this attack the objective of the attacker is to compromise a private key belonging to a private-public key pair used by an Apache server to establish secure communications with its clients, using the *Secure Socket Layer* (SSL) protocol. This attack is another example of serious negative impacts that come from having the snapshot functionality available in the privileged domain.
Acquiring the private key will allow the attacker to impersonate the server to clients because he is now able to sign messages using the server private key and decrypt any messages destined to the server. When the client verifies these messages crafted by the attacker, using the server's public key, the verification is successful and the client is not able to tell if he is talking with the legitimate server or the attacker. The fact that the attacker has the private key in his possession is also propitious to the appearance of powerful *man-in-the-middle* (MitM) attacks.
Another possible attack is eavesdropping or tampering with the communications between the server and the client. For example, let us assume that server and client agreed upon using RSA as the key exchange method for establishing a shared secret key. We are aware that this method has the drawback of trusting the client will generate a strong pre-master secret but, we are assuming this is not a problem and the client generates a strong pre-master secret. When this key exchange method is used, the client generates a 48 byte pre-master secret and encrypts it with the server public key. So, in this situation we would be able to obtain the pre-master secret by decrypting it using the compromised private key. Having possession of the pre-master secret we can then compute the master secret and have access to all the data exchanged between client and server.
Although it is common sense in the security research community that protecting private keys from being captured by attackers is paramount to withhold the security of any system relying on cryptography using such a key, we provide the examples above to clearly illustrate some negative events that might occur if an attacker is able to compromise a private key.

The Attack

Joe decided to use the remainder of his time at INI-Cloud to perform some malicious actions that would for sure, after he left, ruin the image and prestige that INI-Cloud had achieved through the years, in his opinion, at the cost of his hard work. He chose a valuable target, the recent client Cloud-Bank.

Similarly to the first attack, the key insight to reason about this attack is to know that a cryptographic key that is used to perform encryption in a machine has to be available in memory in order to be fetched and used in the encryption operation when required. In our scenario, we did not configure the unprivileged domain to launch the Apache web server at startup. We launched the web server manually to confirm that the private key was only seen in memory when we launched the Apache web server process. It is interesting to observe that we did not perform any connection to the web server through the HTTPS port prior to performing the memory dump that contained the private key unprotected. This means that the Apache process unprotects the key when it receives the passphrase (i.e., the secret that is used to encrypt the private key used by the web server) and then loads it in clear DER-encoding to memory. Later in the text, we provide a detailed explanation of the DER-encoding.

Like we described in the previous attack, it is pretty easy for an administrator to obtain a memory dump from a specific unprivileged domain. In this attack we also need to perform that operation in order to capture a memory image that contains the private key in DER-encoding. After obtaining the memory dump, the attacker needs a tool to extract that information and then he can use it as he sees fit. This is not an issue researchers have come up with various forms to extract private keys from memory dumps. Recently, a team at Princeton University developed a tool [28] to capture keys for the Advanced Encryption Standard (AES) and RSA algorithms.

This was exactly what Joe needed to conclude is attack against the Cloud-Bank private key. After downloading and compiling the tool to retrieve cryptographic keys, Joe ran it against the memory dump file he obtained earlier and, there it is, he gets what he was looking for, the private key used by the Home Banking web server of Cloud-Bank. The attack is complete. In our scenario we were attacking a RSA private key. The RSA algorithm is a public key cryptography algorithm that has as foundation of its security strength the infeasibility to determine the *private exponent*, usually referrer as *d,* when the attacker knows the public exponent and the modulus [40]. This property of RSA is achieve for large values of public exponent and modulus, which in literature are referred to as *e* and *n*, respectively. Obtaining the private exponent violates this property, so the security of RSA is broken. From this point onwards, Joe only needs to use the private key to mount whatever attack

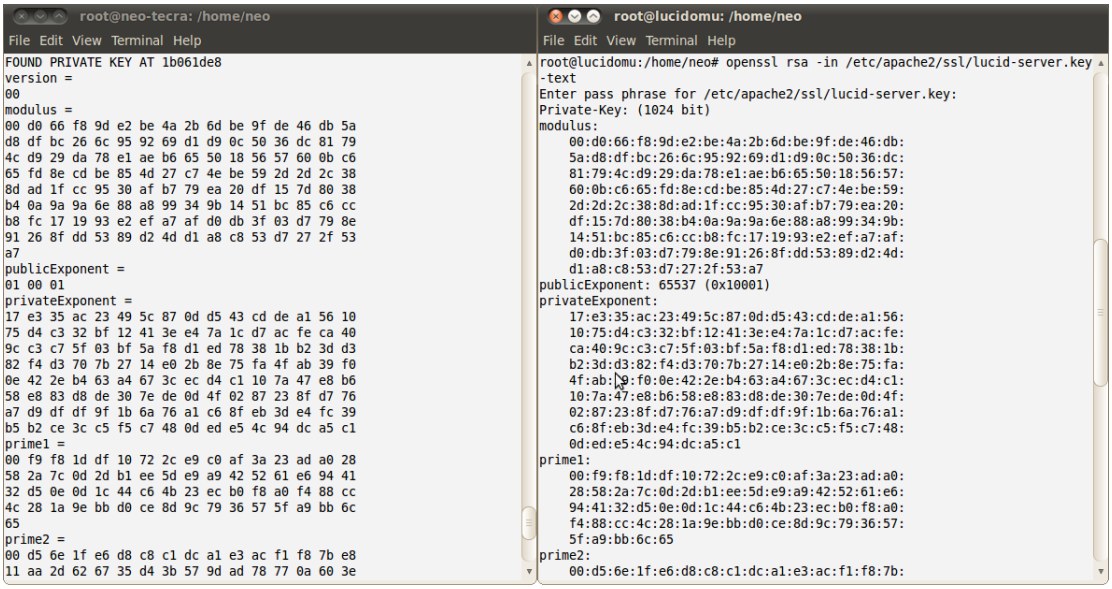

Figure 3.3: Apache web server private key compromised

he wishes to execute against the secure communication channel established between the Cloud-Bank Home Banking server and its clients.

Figure 3.3 shows the steps required to perform this attack against an unprivileged domain. We have also a video recording showing every step taken to extract the cryptographic material from a live memory dump, which is easily obtained as we have already shown in Figure 3.1, from an unprivileged domain [59].

The illustration exhibited in Figure 3.3 displays the outcome of running the *rsakeyfind* tool [28] against the live memory dump we had previously obtained. On the left-hand side of the picture, we have the Linux terminal where we can see the attacker using the *rsakeyfind* tool to extract the private key, the display contains the key we were trying to compromise. On the right-hand side of the figure, we have a terminal window at the unprivileged domain to aid us in confirming that we, in fact, extracted the desired key. From this image we can easily conclude that the private key was compromised by the attacker operating on the left-side terminal window.

The *rsakeyfind* tool uses a novel search method proposed by researchers at Princeton University [29]. Subsequently, we discuss how an RSA private key is represented in memory and how their technique operates in order to extract the private key from a memory dump.

The most widely used form for RSA private key representation is the one specified in PKCS #12, which can be found at [30] or [41]. However, its representation syntax comes from the *OSI networking and system aspects - Abstract Syntax Notation One* or *ASN.1* [42], defined in a four part standard named ITU-T X.680 or ISO 8824. This standard has the

```
RSAPrivateKey ::= SEQUENCE {
    version           Version,
    modulus           INTEGER,  -- n
    publicExponent    INTEGER,  -- e
    privateExponent   INTEGER,  -- d
    prime1            INTEGER,  -- p
    prime2            INTEGER,  -- q
    exponent1         INTEGER,  -- d mod (p-1)
    exponent2         INTEGER,  -- d mod (q-1)
    coefficient       INTEGER,  -- (inverse of q) mod p
    otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
```

Figure 3.4: RSAPrivateKey ASN.1 type

objective of defining standard notation for the definition of data types and values. According to the standard a data type is a generic category of information (e.g., numeric or textual), whereas a data value is an instance of a particular data type. The ASN.1 notation is supplemented with a set of *encoding rules* that specify the value of the octets that carry application semantics, also called the *transfer syntax.* The *encoding rules* used to represent the abstract objects in binary form are described in ITU-T X.690 or ISO 8825, and are denominated *Basic Encoding Rules (BER), Canonical Encoding Rules (CER)* and *Distinguished Encoding Rules (DER)*. The CER and DER definitions are subsets of BER and differ from each other in a set of restrictions.

The general rules for encoding can be found in ITU-T X.690 or ISO-8825-1 [43]. In this document we can read that the encoding of a data value should have four distinct components, namely *identifier octets*, *length octets*, *contents octets* and *end-of-contents octets*, and they must appear in the order they are listed. The important component to distinguish here is the *identifier octets* which is going to encode the ASN.1 tag of the type of the data value. A list of such tags can be found in [44]. For example, the tag used for an integer value is, in hexadecimal, 02. The *identifier octets* byte is always used as the starting byte of any ASN.1 encoding. This byte is divided into three parts: the two-bit classification, the constructed bit, and the primitive type.

The ASN.1 object identifier for an RSA private key can be found in [30]. The document defines object identifiers for both private and public RSA keys, but we are only interested in discussing the private key representation. An RSA private key should be represented using the *RSAPrivateKey* ASN.1 type, that is define as it can be seen in Figure 3.4.

In [29] the authors discuss some previously used techniques to extract private keys from memory and present their method. Their key search method starts by looking for identifying features of DER encoding. They argue that their technique generated no false positive. We were not concern with testing this technique. In our case it was also effective and no false positives occurred. The tool starts by looking for the ASN.1 SEQUENCE type whose uni-

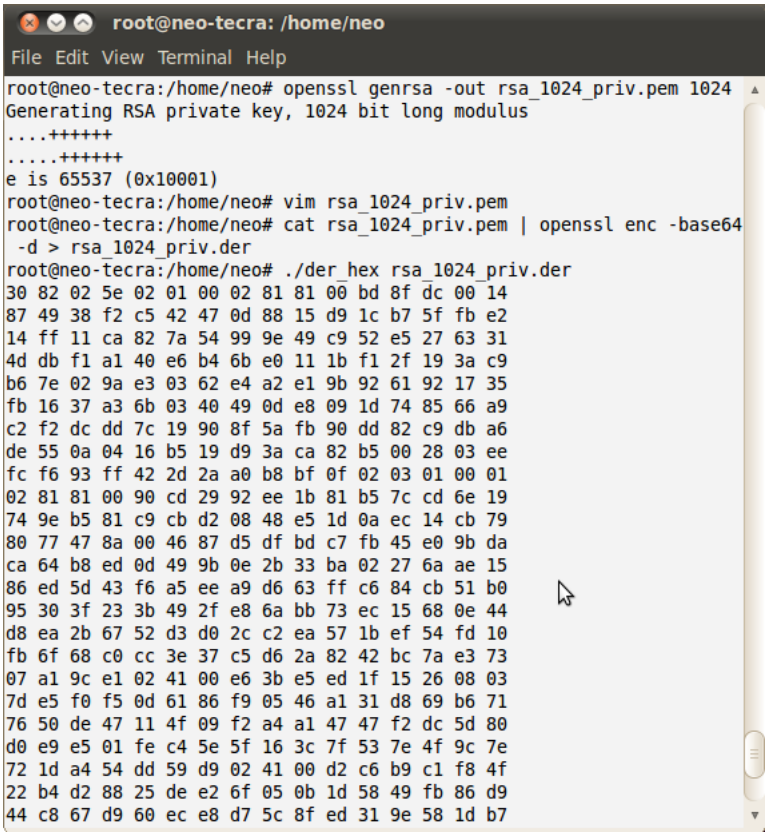

Figure 3.5: RSA private key according to ASN.1 type displayed in hexadecimal

versal class tag is *0x10*, but since, according to X.690, its encoding must be constructed, the SEQUENCE header byte changes to *0x30* because bit 6, the constructed bit, needs to be active. The next step is to locate the RSA version number, which much be zero unless multi-prime is used, in its DER encoding and, finally, the DER encoding tag for the next field.The value, in its hexadecimal form, for this final set of bytes is *0x02 0x01 0x00 0x02*, where *0x02 0x01 0x00* is the RSA version and *0x02* is the type of the next field. Decomposing the RSA version bytes we can see the *identifier octet (0x02 - INTEGER)*, with one byte of length (*0x01*), and with a value of zero (*0x00*) We also know that the next field is going to be another integer. In Figure 3.5, we provide an illustration where the reader can see an RSA private key organized according to the ASN.1 type specification. We used *openssl* to generate an RSA private key, removed some unnecessary content from the file where the key is stored, decoded the base64 format in order to obtain the key in DER encoding, and we used a small C program, named *der_hex,* to print the DER encoded key in hexadecimal. In the picture it is possible to identify the SEQUENCE type and the set of bytes composed by the RSA version and type and length of the next field.

This section is a clear example of how easy it can be for a well motivated malicious admi-

nistrator to breach the security of the cloud infrastructure of a company. The attacker does not even need to be an expert in Linux, he can achieve the described outcome by simply knowing how to interact with Xen and a few Linux terminal commands. A security-educated reader may argue that an integrity-protected hypervisor could prevent this attack, but it is not that linear to avoid this scenario. An attack that is presented later in Section 3.4 of this chapter is going to provide a strong argument in favour of the attacker being able to get an unprivileged domain to execute on a server using a raw and complete installation of a hypervisor. In this scenario, even if the cloud infrastructure has some highly secure machines, the attacker can always redirect the target virtual machine forcing it to launch in an insecure box. We will later discuss this topic in depth.

3.3 Extracting private data from the hard disk

Specific Attack Model

For this attack, we are assuming that cloud users are now aware of the dangers the snapshot functionality represent to their confidential data and have set up a policy where they only consider exporting their private data to cloud computing providers with hypervisors that have the snapshot functionality turned off by default.
We are assuming that the cloud provider establishes a service level agreement with his clients, where he clearly states that all machines in his infrastructure do not have the snapshot functionality and he, in fact, honors that compromise.
We are not considering system like Wuala [45], which provide security to information the user stores in the Cloud, but that information must be encrypted prior to being uploaded into the cloud infrastructure. We are interested in compromising information that is going to be used in the cloud infrastructure.

The attack we describe in this subsection has different outcomes and objectives when compared to the ones previously presented. In the previous attacks we were attacking the contents of volatile random access memory and, here, we are interested in obtaining data kept in permanent storage media, such as the hard disks, used by the victim unprivileged domain. The video we recorded for this attack can be located in [60].
Like we have already done before, we are going to provide two hypothetical scenarios in order to illustrate how this attack can be used to violate the privacy of a cloud infrastructure user. Unlike the previous attack this one has an immediately visible impact on the confidentiality of private data delegated by the cloud user to the cloud infrastructure provider, in terms of contents the cloud user is aware to have stored in the cloud infrastructure. The

information we extracted in previous attacks might not be seen as possible to compromise by regular cloud users. In this scenario the user copies some files he owns to the infrastructure.
If an attacker has the ability to extract contents from the permanent storage media used by a cloud user, the attacker can, for example, extract some data that enables him to gain a commercial advantage over the owner of the compromised confidential data. For example, let us assume that two massive multiplayer online gaming providers, e.g., Dark Throne [31], are using the same cloud infrastructure provider. One of these cloud users could present an interesting monetary reward to an administrator of the cloud infrastructure provider in order to obtain the code of the online gaming engine used by its direct competitor. This would constitute an attack against intellectual property that is owned by the original developers.
In a second scenario, this attack could be used to mount powerful phishing attacks. Let us imagine that the attacker wants to reproduce a certain web application to mount the phishing attack. If that web application requires a secure connection between client and server, the attacker could start by compromising the private key file and then extract the web application source files to have a perfect copy for his phishing site. Here we are assuming that the private key file is not protected with a passphrase, which is an option. In the event of the private key file being encrypted a third attack combination is also possible. The attacker cloud start by getting a live memory dump from the victim unprivileged domain and compromise the passphrase protecting the private key. This attack is possible as we will later describe in Section 3.4. Obtaining this passphrase would allow the attacker to extract the private key file from the victim unprivileged domain even if the key contained in the file is encrypted. The attacker would still need to worry about some details but the attack would have a high probability of being highly successful.

The Attack

Joe was interested in a high profile attack that would definitely get Fortune 100 companies away from the cloud services or any other services offered by INI-Cloud.
Joe started thinking about the attack. He soon realized that like volatile random access memory, the permanent storage media was also being shared between privileged domain and unprivileged domains. Since he did not have any tools to help with obtaining a live memory dump from an unprivileged domain, he thought to himself that there must be a method he could use to make a backup copy of the logical volume used by the unprivileged domain owned by Cloud-Bank.
He knew that the partitioning system they were currently using is the logical volume manager solution or simply LVM. All he needed to find out was how to make a backup copy

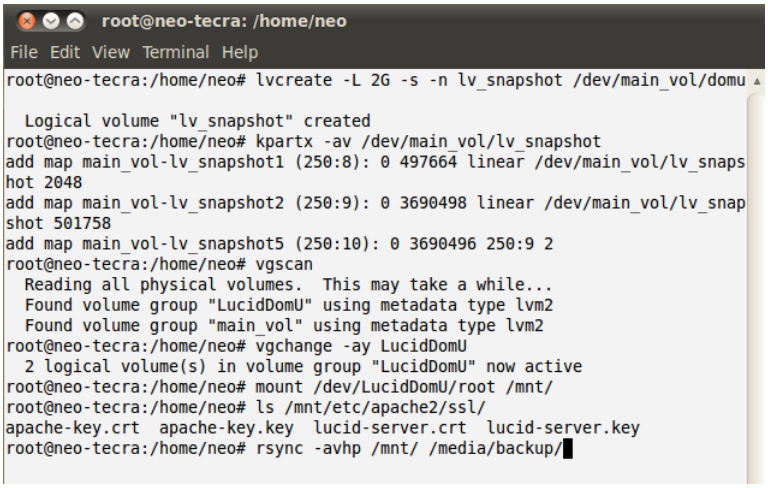

Figure 3.6: Backup an unprivileged domain LVM partition

of an unprivileged domain's partition. After a few queries inserted in his favourite Internet search engine, he came across some information that allowed him to perform the required steps to achieve his final goal. Using that information he was able to compromise the whole content of the file system used in the unprivileged domain belonging to Cloud-Bank. He could now explore the content to attack clients of the home banking solution offered by Cloud-Bank.

The steps taken to perform a full copy of the permanent storage media used by Cloud-Bank unprivileged domain are illustrated in Figures 3.6 and 3.8. These are the steps required to execute the attack we just presented. Following, we have a brief introduction to the LVM approach and then we describe the attack in detail. The majority of the commands that are mentioned in the discussion of the attack, starting with a *lv* or *vg* prefix, are commands related to the *Logical Volume Manager* or *LVM* for the Linux operating system. The LVM solution for Linux machines allows the creation of logical volumes from the permanent physical storage resources used in a machine. The big advantage of using logical volumes instead of physical volumes, is that logical volumes can be expanded or shrunk as needed while the system is running. This capability provides a great deal of flexibility when dealing with the management of storage resources in Linux environments [33].

An LVM environment involves three different levels of abstraction. The first one is where we have the physical storage resources that are going to be used for LVM, these resources are called *physical volumes*. The next level is denominated the *volume group*, which is no more then a set of one or more physical volumes. The highest layer are the *logical volumes*, this is where we can mount or create filesystems. We provide an illustration of these levels in Figure 3.7.

In Figure 3.6 we illustrate the set of commands required to obtain an integral copy of the

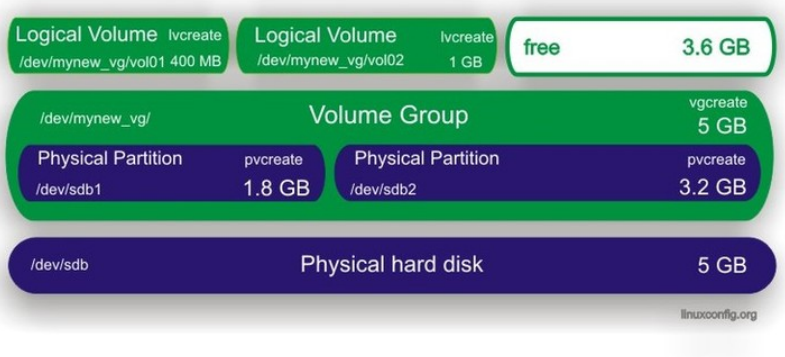

Figure 3.7: Logical Volume Manager (LVM) levels of abstraction [57]

contents stored in a Logical Volume Manager logical volume. The *lvcreate* command entry is used to create a new logical volume that consists of a snapshot of an already existing volume. In detail, we are creating a snapshot of a logical volume, with a size of 2GB, with the name *lv_snapshot* from the original logical volume */dev/main_vol/domu*. As we can see in Figure 3.6 the snapshot volume was successfully created.

Our second step is using the *kpartx* utility to add the partition mappings from the partition table existent on the newly created *lv_snapshot* volume. The *kpartx,* derived from the *partx* command, utility is responsible for reading partition tables existent on specified devices and then create the respective device maps corresponding to its findings. In Figure 3.6, we can see that three new map entries are added.

We now perform a scan through the devices on the system to find all available LVM physical volumes and volume groups using the *vgscan* command. The purpose of the *vgscan* command is simply to scan disks in order to find existent volume groups. The command output indicates that two volume groups, *LucidDomU* and *main_vol*, were found after scanning all physical volumes. The one we are interested in is the volume of the unprivileged domain, as the name clearly suggests, it is the *LucidDomU* volume group. After having the volume group available we move on to activating the logical volumes existent within the volume group in question. We use the *vgchange* command to achieve that objective. Running the command as depicted in Figure 3.6 successfully activates the two logical volumes that exist within the *LucidDomU* volume group.

Now that we have the logical volumes active, we can mount the *root* logical volume in the */mnt* directory and list the content of an interesting directory residing in the logical volume we have just guaranteed access to. From the listing we can see the certificate and key file used by the Apache web server.

Finally, we use the *rsync* file copying utility to make an integral copy of the contents in the root logical volume, of the *LucidDomU* volume group, to a local directory properly named

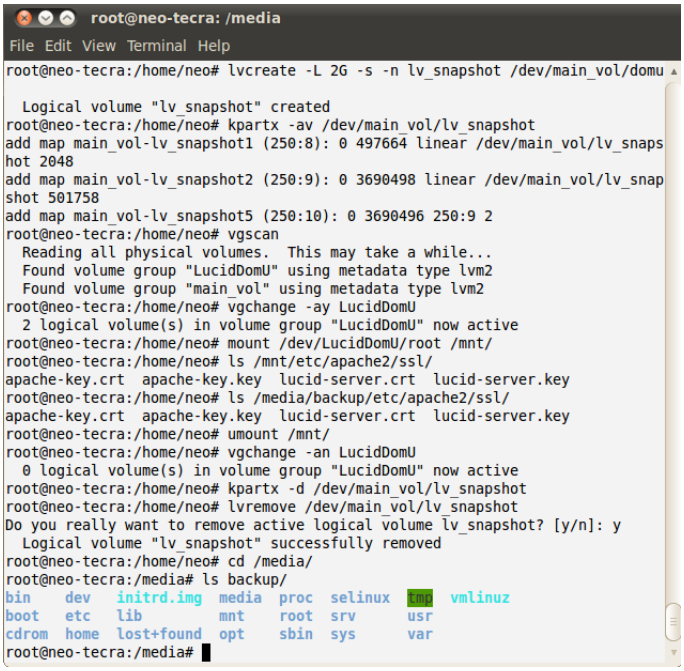

Figure 3.8: Overview of the complete data copy process

/media/backup. The *rsync* tool is a simple local or remote file copying utility. We are interested in doing this because it gives us a copy of the file system of the unprivileged domain ready to be explored. From this point on, any information stored in the unprivileged domain storage space is in the possession of the attacker.

Figure 3.8 gives us a general view of all the process displaying the list of acquired data, the whole file system of the targeted unprivileged domain, as its final command entry. We can also see the steps taken after the *rsync* file copying utility is done copying the content of the target volume. We start by un-mounting the root volume that belongs to the *LucidDomU* volume group. Second, we deactivate the logical volumes that belong to the *LucidDomU* volume group that we had activated in early steps in order to obtain access to its content. The *vgchange* command completes this task. Finally, we remove the partition mappings with the *kpartx* command and remove the snapshot logical volume using the *lvremove* command.

It is intelligible from the illustrations that it is fairly easy for a malicious determined administrator to obtain a copy of confidential data of a cloud user. This attack uses only legitimate Linux commands and is not that complex to perform. High-value businesses will for sure require some kind of backup system to protect their valuable information. This attack shows that the tools required to provide such a backup system might be maliciously used by an ill-intentioned administrator. The results of such use might be considered drastic. The malicious administrator just obtained access to the whole content of the root logical volume

of an unprivileged domain. It is also interesting to notice that a *volume group* is composed by one or more *physical volumes*. Let us assume that an infrastructure is configured to have all its storage resources managed in a single volume group, this configuration would give an administrator access to data store in any of the logical volumes contained within the volume group. From the facts we have exposed we can infer that there are some risks associated to the sharing of storage media among virtual machines.

3.4 Virtual Machine Relocation

Specific Attack Model

The existing security level in this scenario is the highest level of those considered within this chapter. Here the cloud providers take security seriously and have all the machines in their infrastructure equipped with last generation technology. The important element to consider is the Trusted Platform Module (TPM) from the Trusted Computing Group (TCG) [20]. In theory, the security properties offered by the TPM hardware module have a considerable impact in the security level of the cloud infrastructure.
One of the mechanisms provided by the TPM is the ability to present to the client an attestation assuring that the running hypervisor is indeed an integrity-protected hypervisor. This guarantees the client that he can trust the operations executed by that hypervisor. Good examples on how the security of a hypervisor can be improved using these approaches can be found in [14] and [13].
We are assuming that the machines equipped with TPM hardware modules and running integrity-protected hypervisors require network capacity using for example SSH, or any other form of remote communication, so they can schedule virtual machines to be launched in different servers or else the advantages of cloud computing would be lost. Operating in a single machine would not provide any advantage to the cloud user and would definitely create availability problems.

In this subsection we are going to present a considerably different and relevant attack. This attack has that elevated importance because it can be the key to subvert an otherwise seemingly highly secure system, and turn it into a system where the attacker can perform all the attacks we have previously described. A video recording can be seen in [61]. It is also worth to mention that the attack is important because in a system that is considered highly secure the administrators are usually more confident in their technology, and that might ease the task of a determined attacker. This attack has been described previously in [32], so our main contribution here is only to show it working and to demonstrate in yet another

way the privacy issues of cloud computing.

This attack scenario can be divided in two main parts. In the initial phase of the attack, the objective of the malicious administrator is to assure to the cloud user that the machine where he is going to launch the client's virtual machine is using an integrity-protected hypervisor. The attacker will do this using an attestation mechanism. This attestation is possible due to the TPM hardware module present in the machines operating in the cloud infrastructure. In our case, we are going to demonstrate the attack using a direct proof mechanism based in the code found in web page of PrivacyCA [34].

When the attestation process is complete and the verifier trusts the hypervisor running on the cloud infrastructure, the attacker can simply relocate the virtual machine he is about to launch to a completely different physical machine. This machine would be running a neither hardened nor integrity-protected version of the hypervisor and privileged domain, offering resources to the attacker that can help him with the process of subverting the privacy of the infrastructure. For example, the snapshot subcommand of the Xen management user interface already discussed in this chapter. To prevent this relocation we would also have to attest the integrity of the virtualization management software, and even then, the attack might not be stoppable with current technology [13].

The ability to mount this attack will give the malicious administrator a green card to perform any of the attacks previously discussed. This attack would render any attested integrity-protected hypervisor irrelevant from a security point of view. An attacker could then mount a very powerful attack against the privacy of the cloud user. The attacker could start by obtaining all the information he could from live memory dumps obtained from the victim unprivileged domain. This would allow the attacker to extract, for example, system passwords or passphrases and private keys. He could later extract all the information from the hard disk and literally, at this stage, he would have managed to replicate the unprivileged domain operating at a certain point in time.

The ability to collect all this information clearly assures that the privacy of the cloud user can be violated. From this point onwards, the attacker could target the clients of the cloud user he just attacked and mount very powerful attacks against those clients.

A good example would be the almost perfect phishing attack where the attacker is capable of cloning the web application and obtain the private key used to establish secure connections to the original application. This would allow him to have the right URL in the certificate details, and he would only have to worry about crafting an intelligent email message to make it seem the legitimate application even though the URL showing in the browser is different. This attack would definitely have a high success rate among security unaware users. At this stage, other powerful attacks can be orchestrated against the confidentiality and integrity of secure communications between clients and the legitimate web

applications resident in the cloud infrastructure, the attacker has enough information in his position to mount attacks against the system.These are just short examples of what could be achieved by subverting the privacy of the cloud user. New attacks would be dependent on the creativity and objectives of the attackers.

The Attack

Joe was impressed with the new security measures a recently hired INI graduate had set up in the cloud infrastructure of INI-Cloud. This integrity-protected hypervisors and attestation mechanisms offered by the Trusted Platform Module (TPM) were a considerable challenge to overcome but Joe was determined to pull through. He started learning about the details of trusted computing to combine them with his thorough knowledge of cloud computing, so he could find a strategy to circumvent the existing protection mechanisms.
After a few days of intensive research and dedication Joe realized that the attestation was regarding a single machine and that he could use a relocation strategy to launch the virtual machine in a different machine operating with an hypervisor he could use to attack the unprivileged domain. He found that this relocation was possible because the virtualization management software was not included in the trusted computing base to avoid bloating its size. From this point on it was easy, Joe simply needed to guarantee that the cloud user received an attestation of an integrity-protected hypervisor running in one of the machines and then he relocated the launch operation to a different machine running a hypervisor and privileged domain he had compiled and installed.
When he managed to launch the victim virtual machine in a machine that was not running an integrity-protected hypervisor he could mount privacy attacks against confidential data residing in main memory and logical volumes reserved for the victim unprivileged domain.
The relocation vulnerability was first discussed in [32]. That paper argues that trusted platforms can reliably detect whether or not a single host is running a trusted platform implementation. This assures that they might effectively secure a virtual machine running on a single host. The problem arises when the infrastructure is composed of several hundreds of machines and the virtual machines can be dynamically scheduled to run on any of them.
In Figures 3.9 and 3.10 we can see an illustration of the attack. Subsequently, we are also going to provide the reader with a detailed description of the attack demonstration we simulated in our test environment. As we have already mentioned, we used the source code provided at PrivacyCA's web page [34] as a starting point in order to assemble a working version for our specific environment conditions.
In Figure 3.9 we can see the steps taken in a secure machine while it is being challenged by a remote verifier. The steps required to achieve a direct attestation are all visible in this image. First, the certificate chain is used in order to generate an attestation identity

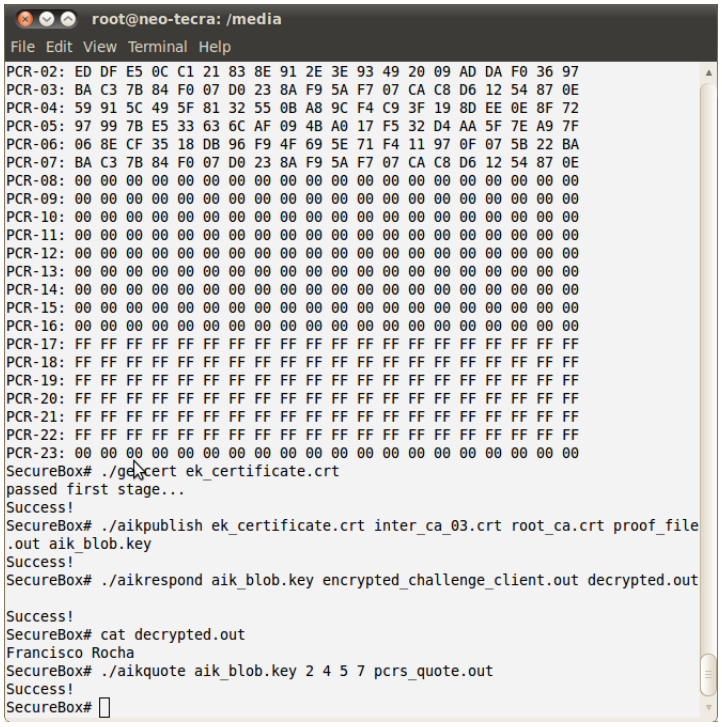

Figure 3.9: Command line view at the secure machine

key and a proof file, which will be used during the attestation process. This is when we call the *aikpublish* application with the required arguments, which include the certificate files and the output files for the proof and AIK. Second, the secure system responds to a challenge the client encrypted using the proof file published by the secure system as an output in the first step. The application named *aikrespond* is responsible for this second step, it receives the AIK, encrypted challenge and name of the file where it should save the result of decrypting the challenge. Finally, the secure system produces a quote of some of its platform configuration registers, so it can attest to the verifier that it is indeed running an integrity-protected version of a hypervisor. The line where this takes place is when we call the *aikquote* application passing it the AIK key, the number of the PCRs we want to include in the quote and the output file where it is going to store the output quote.

In Figure 3.10 we can analyse the steps taken at the verifier during the attestation process between the two machines. The verifier starts by creating the encrypted challenge for the secure system, while at the same time it obtains a copy of the public attestation identity key from the remote system it is trying to verify. The line where we execute the *aikchallenge* application is when we request that operations from the machine. The application receives the as arguments the secret or challenge in plaintext, the proof file from the provider and, the names of the output files where it stores the encrypted challenge and public AIK. After this step it uses that public attestation identity key to verify the quote of the platform

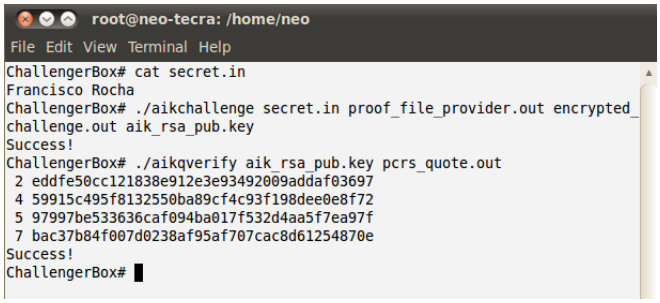

Figure 3.10: Terminal at the challenger box

configuration registers sent to it by the secure system. This takes place when the *aikqverify* application is executed. The arguments for this application are the public AIK and the quote from the provider.

We left the registers in question visible in Figure 3.9. If we compare them side-by-side with the output of *aikqverify* in Figure 3.10 we can easily determine that the applications used to create the direct attestation functionality are correct when they qualify the system as trustworthy. At this point in the process the verifier believes that its virtual machine is going to be launched in a secure environment and is for that reason satisfied with the proof presented by the cloud infrastructure provider.

The problem arises when a malicious administrator manipulates this process and when the virtual machine is about to be launched in the secure machine he suddenly decides to redirect that launching operation to another machine that is not running the integrity-protected version of the hypervisor. In our illustration of the malicious actions that compose this attack, we use SSH communication to establish a link with a rogue machine in which we wish to launch our victim virtual machine. This is just a simple solution to illustrate the steps an attacker would go through to achieve his final objective. Those steps are depicted in Figure 3.11.

In Figure 3.11 we can clearly see that the victim virtual machine ends up running in a different server than the one it ran the direct attestation protocol with. We can see in the terminal that the attacker establishes a SSH connection with a machine with an IP address equal to 192.168.1.69, which is not the one the verifier performed the attestation with. The command to launch the unprivileged domain is the *xm create*. This will definitely allow a malicious administrator to compromise the privacy of the redirected virtual machine because any of the attacks that were previously describe are now possible to execute against the victim virtual machine. This problem is pertinent because we cannot assure the integrity of the virtualization management software, in this scenario we are only providing an attestation on behalf of the integrity-protected hypervisor. Research regarding this specific issues demonstrates that the inclusion of extra components in the integrity-protected do-

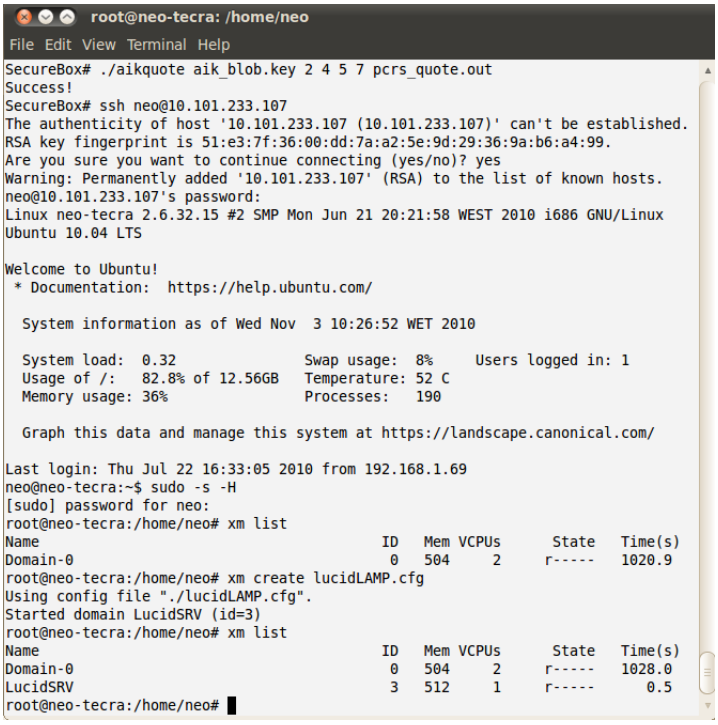

Figure 3.11: Virtual machine running in an insecure machine

main are going to increase the size of the trusted computing base [12, 14] and that is not positive for the attestation of the system.

3.5 Using the hypervisor to monitor executing binaries

This subsection is dedicated to a family of attacks that we did not implement, but that it makes sense to discuss. Subsequently, we expose the reasons to why we chose not to follow this possible venue of attack.

In Section 2.3.2.2, we have already discussed the various types of introspection techniques used in virtualization with security objectives. We have also found literature [46] focusing on using such techniques to monitor the cloud user in order to detect signs of malicious activity on their virtual machines, but they can also be used with the almost opposite objective: for a malicious administrator to obtain illegitimately information of the virtual machine operation. An important point to discuss is the capabilities of introspection techniques. The authors provide a table displaying a classification to each technique in terms of *power*, *unintrusiveness* and *robustness*. According to the authors, *power* is the measure of the scope of VM events an approach can monitor and its ability to interpose on specific events. *Unintrusiveness* measures the level of disturbance an approach imposes in the monitored

	Power	Unintrusive-ness	Robust-ness
Host agent	Good	Poor	Good
Host agent w/ driver	Best	Worst	Poor
Trap/Inspect	Best	Good	Worst
Checkpoint/Rollback	Best	Good	Poor
Architectural	Poor(?)	Good	Best

Figure 3.12: Capabilities of introspection techniques [47]

VM and, the *robustness* of an approach is dependent on the nature of the assumptions made about the monitored VMs and how likely these assumptions are to hold. We show a table contained in the paper in Figure 3.12. This figure presents how good the individual introspection techniques behave for each classification property.

In our context, we are interested in how a technique behaves in terms of *power* because we need to capture as much information as possible, the behaviour of the introspection approach in terms of *unintrusiveness* is also relevant due to the fact that we want our attacks not to be noticed by the victim VM, and robustness is also interesting since we do not want to be dependent on any assumption of the system operating on the victim VM.

The *host-based* approaches (e.g., inserting a module in the VM's kernel) are of no use to us. Like it is mentioned in the article this is the most intrusive type of introspection, it violates the boundary between the provider's and user's realm. A security aware cloud user would not agree to install a monitoring tool in is virtual machines knowing that the provider could use it to compromise his privacy. This might get even more difficult in current times due to the enormous amount of discussion surrounding the privacy of Internet users.

Trap and inspect, and checkpoint and rollback, techniques are both powerful approaches but, the first approach has the problem of being very complex to implement, while the second one is easier but relies on injecting code in the monitored VM in order to use functionality provided by the OS to inspect it. They are not that useful to us because we are not interested in interfering with the victim VM, and they also exhibit some deficiency in terms of robustness.

Taking into account the classification approach proposed by the authors, the architectural technique would be the one that interests us the most because it is robust and unintrusive at the same time. This happens because it only monitors stable, low-level interfaces through the hypervisor. For example, it may depend on inspecting the instructions executed by the processor or the executable file formats. These are OS-indepent interfaces, the authors in [46] name them OS-*agnostic*. However, these techniques require a lot of changes to the hypervisor in order to be implement and impose an overhead in the exe-

cution time of applications running in the monitored virtual machine. An example tool is *patagonix* [47], which was projected to detect and identify covertly executing binaries without making any assumptions about the OS kernel. To detect code execution patagonix depends only on the processor hardware and to identify code and verify code modifications it is dependent on the binary format specifications of executables. The tool was developed with the purpose of detecting and removing malware and rootkits, but is clear that by being able to identify what is running on a monitored virtual machine it can be used to violate the privacy of a cloud user.

We considered a possible attack using this approach. The attack consisted in understanding in detail the code of an application that makes use of private keys e.g., Apache, so we could then use the acquired knowledge to try capturing the set of instructions that could help us in extracting the private key. This attack would have a similar impact to the one we presented in Section 3.2 without the need to obtain a memory dump. Like we have mentioned in this paragraph, this type of approach has two main problems. First, it requires changes to the base hypervisor and that would not be possible when we are using an integrity-protected hypervisor. The addition of these capabilities would not be permitted by cloud users in such type of hypervisor. Second, it introduces overhead in executing applications. The monitored virtual machine can also take advantage of the existent overhead to detect the malicious behaviour from the cloud provider's hypervisor. This limitation is important because it would make this type of attack detectable in current cloud solutions. These are the main reasons to just discuss this type of attack and not execute it.

3.6 Summary and Discussion

It is important to recall that the main focus of our work is in finding arguments to whether or not the privacy of the cloud user can be kept once he relinquishes control over the underlaying infrastructure that executes his sensitive data within the cloud. We are not interested in producing powerful automated attack tools to be used against the cloud infrastructure but instead find its weak points, so they can be corrected in a near future. We limit our demonstrations to proving that it is in fact possible to mount a specific attack against the cloud infrastructure.

The reader may argue that it might not be correct to simplify an attack against a secure machine that is running an integrity-protected hypervisor and using a TPM hardware module to provide an attestation on behalf of the legitimate software. However, recent research go as far as concluding that in practice it is not possible to realize an integrity-protected hypervisor on current x86 hardware platforms [13]. With this in mind we consider that there

is still much work to be done regarding this specific topic.
Finally, based on the information we provide in this chapter, we can say that currently there are no mechanisms that allow a cloud infrastructure provider to present to his clients an accurate report proving how secure it is to use his cloud infrastructure. It is our belief that if a Joe exists out there working for a cloud provider, which is in a vulnerable and delicate position. We will provide a more thorough analyses of current and future state of security and privacy in the cloud infrastructure when appropriate.

Chapter 4

Protection Mechanisms

This chapter presents a detailed study of recent protection mechanisms designed or applicable to securing the cloud infrastructure with the purpose of showing that they do not solve the cloud privacy problem. Some of these protection mechanisms can be seen as a solution trying to solve the whole privacy problem inherent to Cloud computing [32], whereas others are only mechanisms that could be used as a part of a more complete solution [35].

We present a standard structure for each mechanism. Each analysis starts with a description of the mechanism in question, where we briefly describe the mechanism in such a way that the reader can understand the gist of the idea behind it. After apprehending the essence of the solution we move forward by discussing the security properties offered in terms of *confidentiality*, *integrity* and *availability*. Our intention with this section is to give an idea of what the mechanism can offer in terms of security to the cloud infrastructure. Finally, we analyze how well the particular mechanism withstands the attacks shown in the previous chapter.

4.1 vTPM: Virtualizing the Trusted Platform Module

4.1.1 Description

In [35], the authors argue that a single hardware trusted platform module (TPM) is not enough to achieve the desired security properties in a virtualized environment and propose the use of a virtualized TPM (vTPM) as a solution to the problem. They explain that every virtual machine running on the platform with the need to use TPM functionality should be made to feel that it has access to its own private TPM, even though there may be more

virtual machines than hardware TPMs on the system.
They propose a solution that offers to a virtual machine the same usage model and TPM command set to an operating system (OS) running over a VM, as a physical TPM provides to an OS running directly over the hardware platform. It must also maintain a strong association between virtual machine and its vTPM through the life cycle of virtual machines. Another strong association needs to be maintained between the vTPM and its underlying trusted computing base (TCB). Finally, they want a vTPM to be clearly distinguishable from a physical TPM because of the different security properties they offer. The text discusses two possible implementations, a software implementation and another one using a secure coprocessor. The authors implemented a prototype using the Xen hypervisor.

4.1.2 Security Properties

This is no more than an extension of the TPM functionality to support the particular characteristics of the virtualization environment. Like a physical TPM solution the objective of the authors is to offer binary measurements of the running environment to provide remote attestation to a challenging system. The main outcome ought to be the assurance that the integrity of running software has not been compromised.
From a security perspective, the main challenges to the vTPM approach are:

- Unmodified TPM usage model.
- Strong Virtual Machine to vTPM association.
- Strong association of the vTPM with the underlying TCB.
- How to guarantee challengers that the vTPM is capable of providing attestations of its mutable trusted computing base.
- The reestablishment of trust when the vTPM instance is migrated with its virtual machine to a different platform.

Including the vTPM code in the trusted computing base increases the size of the TCB, which is not good from a security perspective [12]. This might pose a serious issue if we are considering the use of vTPM to assemble a secure cloud infrastructure.

Confidentiality Assuming a sealing operation is not breakable by an attacker, vTPM technology offers data confidentiality using the sealed storage operation. In order to assure this property, we also assume the security of the storage root key, or root of trust for storage, is guaranteed.

Integrity The binary measurements offered by a vTPM can attest to the integrity of running code, assuming a verifiable trusted computing base and the security of the root of trust for reporting.

Availability Authors mention that some availability issues might arise in the process of migrating virtual machines. During this process, malicious software might try to alter or omit state.

4.1.3 Protection against our attacks

A recent paper [36] considers that trusted computing on its own is not a solution for all the security challenges we can encounter in Cloud Computing. Since vTPM solely has the objective of porting the security properties offered by the TPM to the virtualization environment it is not going to be a complete solution on its own. Despite all this, the vTPM approach already has two known vulnerabilities as we can learn from [14]. The first one arises from the lack of communication between the libxc domain builder and the vTPM software. This question makes the implementation of this solution vulnerable to a time-of-check-time-of-use (TOCTOU) attack. A detailed description of the attack can be found in the article in question.

The second problem comes from one of the main security concerns aforementioned, which is the ability to assure the integrity of a constantly mutating underlying trusted computing base. This does not only make the size of the TCB unpredictable but, it also bloats its size. As already mentioned in [12] and in this text, this fact is clearly not advantageous from a security point of view. The attacker can use this property to mount attacks against the confidentiality of running unprivileged guest virtual machines. For example, the attack we have demonstrated in section 3.2, where we describe and demonstrate how to extract a private key used to establish secure connections between a client and a supposedly secure Apache web server.

Since in our work we are impersonating the malicious administrator role, we can see some possible extra venues for attacks. These windows of attack open when the implementation of vTPM runs outside a secure coprocessor. The authors do mention that it runs as a privileged domain user process, so since we can dump memory, we can attack important processes such as the key generation operations. With this type of attack we could successfully capture the endorsement created for new vTPM instances or the attestation identity keys they subsequently generate.

Another point of concern is that, as a privileged domain malicious administrator, we are also going to have control over a fairly powerful feature, the ability to spawn and generally

manage new vTPM instances. We have not studied in implementation in detail but this could, for example, allow us to generate new vTPM instances for malicious virtual machines.
Considering all the facts we have just discussed, we think it is safe to say that using vTPM to protect against the demonstrated attacks is not an effective solution. Like we have mentioned, the use of trusted computing on its own is not a complete solution, when we had vulnerabilities to the trusted computing layer it becomes pointless to use such mechanisms.

4.2 Private Virtual Infrastructure for Cloud Computing

4.2.1 Description

In [37], the author starts by stating that traditional security methodologies are not going to be effective in the cloud environment. The author then proposes a novel solution based on service level agreements (SLAs) between cloud provider and information owner. In his view, having the information owner control the Private Virtual Infrastructure (PVI) while the service provider controls the cloud fabric is the best model to achieve the desired security properties for this new environment. The main idea behind the PVI datacenter concept is to have a virtual datacenter over the existing physical cloud infrastructure. The PVI architecture has vTPM and Locator Bot (LoBot) as its building blocks to create a secure cloud computing infrastructure environment. The vTPM provides the root of trust while the LoBot protocol allows each virtual machine to be verifiable remotely by its owner.
The PVI factory is a critical point in the infrastructure because if it is compromised the rest of the PVI components are at risk and future provisioned components cannot be trusted. It is in the PVI factory that the vTPM keys and certificates are generated and managed.
The authors present five security requirements for their architecture and name them the five tenets of cloud computing security. The five tenets are:

- Provide a trusted foundation on which to build the private virtual infrastructure. This is accomplished through the service level agreement with the service provider assuring they will provide the requisite security services necessary to protect the information with PVI.
- Provide a secure factory to provision the PVI. The factory also serves as a policy decision point and root authority for the PVI.
- Provide a measurement mechanism to validate the security of the fabric prior to provision of the PVI.

- Provide secure methods for shutdown and destruction of virtual devices in the PVI to prevent object reuse attacks.
- Provide continuous monitoring and auditing from within the PVI as well as from outside the PVI with intrusion detection systems and other devices.

4.2.2 Security Properties

This solution proposes no novel security schemes. It simply uses existing security technologies and tries to combine them to compose a secure cloud infrastructure for the information owner or cloud user. Its security properties come from using virtual TPM and Locator Bot as its security foundations.

Confidentiality this property can be achieved through the use of encryption or sealed storage offered by vTPM. The author explicitly mentions the use of SSL tunnels to secure data within the cloud, whether it is communication to or within the PVI.

Integrity The use of binary measurements provided by vTPM is the source of integrity validations. The Locator Bot is the tool used to pre-measure the targeted cloud infrastructure for the desired security properties.

Availability Since the solution uses vTPM as a foundation it is vulnerable to the availability issues mentioned for this technology [35]. An attacker can also try to influence the outcome of the Locator Bot pre-measurement to mount a Denial of Service (DoS) attack.

4.2.3 Protection against our attacks

This paper proposes an interesting solution. Although it does not propose any novel security approaches to secure the cloud, this article is clearly going in the right direction. The author has already reached the conclusion that only a solution composed by various mechanisms is going to be able to offer security to the cloud user. Unfortunately, its root for trust, the vTPM, used at the provider has already been broken. In [14], the author presents a TOCTOU attack against the vTPM as we have mentioned in the previous section.
The main problems of this proposal are that it is based on the assumptions that both parties keep their part of the SLA and, that the PVI factory is not compromised. Another problem is that it uses vTPM as a foundation for its security and, as we have mentioned, this solution is no longer considered secure.

The use of vTPM as its root of trust and the assumption that both parties hold up their part of a service level agreement make this solution vulnerable to attacks orchestrated by a malicious administrator. In our work, we are considering an attack model where a malicious administrator is trying to subvert the system, and since vTPM has already been proven insecure, as a malicious administrator we have some possible targets to subvert the security of the private virtual infrastructure.

The main target would be the private virtual infrastructure factory, which can be attacked if it is using the version of vTPM that is not protect in a secure co-processor. We have already discussed that the vTPM solution as a couple of vulnerabilities that allow a malicious administrator to subvert the confidentiality of unprivileged guest virtual machines.

Another possible target is the SSL tunnels used to assure confidentiality of communications to or within the PVI. We have demonstrated an attack that gives us the ability to extract a private key used to establish SSL tunnels, so we can use a similar attack to obtain the keys used to establish these secure channels.

Another possible attack, although it is not stealthy, is a possible denial of service against this infrastructure. For example, after capturing the keys generated in the vTPM processes, the malicious administrator could influence the outcome of the pre-measurement operation performed by the Locator Bot prior to the deployment of a given private virtual infrastructure. A malicious disgruntled administrator could be the source of such an attack if he is trying to give a bad name to the organization that he sees as the source of his personal problems.

The author never mentions the integrity and dimension of the hypervisor and how is the solution in terms of size of the trusted computing base. These are relevant security aspects [12]. If we consider the various components of the solution, the size of the trusted computing base might be considerable. To assure the good operation of the solution we might have to include the hypervisor, vTPM and Locator Bot code in the trusted computing base. We need to include these components because they are the security foundations for this solution, and if the attacker can compromise them the solution loses its value. We conclude that the solution would also require some work with respect to the size of its trusted computing base, and about which relevant components should be included in it.

Analyzing the five tenets of cloud security proposed by the authors, we conclude that their proposal fails to comply with four of them. The first one fails because the root of trust is vulnerable to attacks as we have already discussed. The second and third tenets are not assured due to failure in complying with the first one. If the root of trust is not secure then any measurements or secure components assured or provided by it are also considered insecure. Assuming the fourth tenet also bases its security in the security properties offered by vTPM, it is also not guaranteed by this solution because of reasons already mentioned.

As a final note, we would like to leave an open discussion topic regarding the amount of information related to the security configuration of the cloud infrastructure that should be shared between cloud provider and cloud user. The author suggests the cloud provider should share a considerable amount of information with the cloud user so they can reach an agreement in their service level agreement. In our opinion, this level of information sharing can be very dangerous. We have seen attacks against the cloud that can be mounted from the outside [10] and this extra information would only make this type of attacks easier. So, it is a delicate subject and the level of shared information needs to be carefully considered.

4.3 Towards Trusted Cloud Computing

4.3.1 Description

The authors propose a Trusted Cloud Computing Platform (TCCP) using the Trusted Platform Module (TPM) as root of trust [32]. The main purpose of TCCP is to provide a closed box environment for the guest Virtual Machine (VM) running in the cloud provider's infrastructure. The closed box abstraction has the objective of keeping a privileged administrator from inspecting or tampering with the contents of running unprivileged domains. The TCCP also offers remote attestation to a client wishing to launch a VM in an environment that is not trusted prior to attestation.
The trusted computing base (TCB) of the TCCP includes two main components: a trusted virtual machine monitor (TVMM), and a trusted coordinator (TC). The trusted coordinator is supposed to be maintained by an external trusted entity (ETE). This external entity is thought of as an equivalent to VeriSign and homologous certification authorities. The TCCP operates by creating a set of trusted nodes, managed by the TC, that are going to be used to run the guest unprivileged domains. It also aims at protecting launch and migration operations, which are critical stages of VM management. Basically, it first establishes trust for the nodes existing within the infrastructure and then uses that trust to create a trusted environment for the guest unprivileged domains.

4.3.2 Security Properties

This solution bases its security in the security properties guaranteed by the TPM. It then builds a trusted environment based on that root of trust, where it can perform secure launch and migration operations with virtual machines.

Confidentiality The solution offers data confidentiality based on the assumption that it assures the integrity of running code. If this assumption fails, data confidentiality is not guaranteed. The approach proposes the use of the endorsement key (EK) as the key to sign on behalf of the TPM. This violates the privacy of the TPM in question and, at the same time, breaks the confidentiality of the internal architecture of the cloud provider by revealing the identity of its machines. Attestation Identity Keys (AIKs) are the ones usually used to sign binary measurements on behalf of the TPM.

Integrity The binary measurements offered by TPM are used to guarantee the integrity of code running on the trusted nodes that are going to be used to run the guest unprivileged domains.

Availability The trusted coordinator is relevant for the availability of this solution. If an attack is mounted against it, the solution will not be able to obtain new trusted nodes and that is going to influence its availability. The addition of an external entity to the secure cloud infrastructure brings this disadvantage with it.

4.3.3 Protection against our attacks

This article definitely presents the work that is closest to what we might be looking for in order to overcome the security challenges inherent to the cloud computing infrastructure. The proposal bases it solution on the root of trust provided by the TPM and tries to use that functionality to establish a set of nodes as trusted nodes to the cloud user.

4.3.3.1 Implementing a trusted virtual machine monitor

Considering its recurrence to trusted computing we can envision this solution as being quite resilient to attacks from a malicious administrator. However, there are some questions that need to be mentioned regarding how practical it is to implement such a solution. The solution is based on a trusted virtual machine monitor or an integrity-protected hypervisor, and recently, a pertinent question regarding the problem of implementing an integrity-protected hypervisor using current x86 technology has been raised [13]. Assuming the information in the article we just mentioned is correct, the ability to provide a TVMM might not be achievable using currently available technology. This would render this solution ineffective from a security point of view because if we cannot assure the integrity of the hypervisor the attacker can use the privileged access to subvert the security of the cloud infrastructure.

4.3.3.2 Forbidding physical access

Although the author assumes that physical access to the servers is not guaranteed to any employee, we do not believe that currently it is feasible to forbid this access. Having access to the machines we still have problems due to the fact that the key used by a registered trusted node is maintained in main memory and it is assumed that after reboot the trusted node is removed because it no longer possesses the key in the volatile memory. This is not quite true with access to the machines, it is possible to mount an attack [29] to extract the key and then reboot the machine with a malicious hypervisor in order to compromise the privacy of unprivileged domains scheduled to run on that machine. Like it is mentioned in [29], there several easy methods to quickly obtain a memory dump. When the memory dump is obtained the malicious administrator only needs to extract the key from it. This attack being possible renders the rest of the solution insecure because every protocol mentioned is based on the compromised key. The confidentiality, integrity and availability properties are no longer assured.

4.3.3.3 Adding a trusted coordinator

We also consider that the addition of a trusted coordinator, maintained by an external trusted entity, adds unnecessary complexity to the problem of securing the cloud infrastructure. A similar problem was already observed with respect to the PrivacyCA issue [25]. This trusted coordinator is going to open other attack venues that can be exploited. For example, if an attacker is able to conceive a viable way of impersonating the trusted coordinator to the IaaS perimeter, the solution is no longer secure. From that point onwards, the attacker can control which nodes are considered trusted nodes. If it is not that, it could also be an availability attack.

Since the cloud infrastructure is supposed to provision resources on-demand, the exhaustion of existing trusted nodes might create a high volume of traffic between the cloud provider and the external trusted entity. This communication link could be targeted by attackers in order to diminish the level of availability of the infrastructure. This attack could be used by a competing cloud provider that uses a different external trusted entity, and by targeting the communication link between the other provider and its external entity the malicious cloud provider would be able to have better availability levels.

4.3.3.4 Final considerations

Another possible attack is mentioned in [10]. There is nothing preventing an ill-intentioned client from launching a malicious VM in a way that it ends up running in the same node as the targeted benign VM.
The fact that the endorsement key is used to register a trusted node is also not a positive point in favour of this solution. As we have mentioned this act violates the privacy of the TPM belonging to the registering trusted node.
Albeit possessing all these currently unsolved issues, we consider this the most secure solution currently available, because it clearly reduces the probability of our attacks taking place. It is going to take a very powerful malicious administrator in order to break this secure cloud infrastructure setting. The problem is that if it gets broken once, it is always going to be broken because the attackers can create automated tools to subvert this specific scheme.

4.4 DepSky

4.4.1 Description

We dedicate this section to a novel cloud storage solution. We have decided to include DepSky [52] in our work because it presents an interesting and promising approach to offer secure storage in the cloud. DepSky is a system which main purpose is to guarantee confidentiality, integrity and availability of information stored in the cloud. The foundation of the DepSky system is the use of replication of information through multiple storage cloud providers, using algorithms to achieve reliable storage and secret sharing. The atomic building block of DepSky are data units, that can only be altered by their owner and read by an arbitrary number of clients or readers. The system uses replication algorithms for quorum Byzantine dissemination systems to assure the availability of these data units even in the presence of failures. The data units are composed by a metadata file and another file containing the most recent version of the data. The contents of the metadata file are a version number, a cryptographic hash of the data and a pointer to the file containing the data for this specific version.
The DepSky system has an availability algorithm denominated ADS, which is responsible for the improvements in the availability of cloud storage services through the replication of data units in multiple storage cloud providers. The algorithm has a writer and reader version, which are used by writing entities and reader entities, respectively. These operations

always require at least *n - f* writing confirmations or *n - f* valid reads. The CADS algorithm or Confidential & Available DepSky, which includes a cryptographic secret sharing algorithm. In this algorithm, the secret stored in the cloud is only obtained after collecting *f + 1* parts of the secret.

4.4.2 Security Properties

The security of the DepSky system relies on quorum Byzantine dissemination system principles, which means that the system is resilient to *f* servers exhibiting Byzantine behaviour, if it has a number of servers equal or greater than *3f+1.*

Confidentiality This property is assured through the use of the secret sharing algorithm. The data stored in the cloud is divided between a group of *n* participants, each of this participants is granted a part of the secret. The whole secret can only be reconstructed when *f + 1* of those parts can be collected by the entity that wants to obtain the secret.

Integrity The metadata file contains a cryptographic hash of the data that is stored in the cloud. Using this information an entity that is reading the data can verify if it has been modified and act accordingly.

Availability The improvement DepSpky proposes to availability comes from using replication to store its *data units* in various store clouds instead of a single one.

4.4.3 Protection against our attacks

This solution does not defend against the attacks we have demonstrated in this work. It can only offer security properties to storage cloud solution. In our case, we are more concerned with general purpose cloud infrastructures, where the information is actively manipulated and modified by cloud servers. DepSpky is only useful when the information is securely stored in the cloud and only the owner can modified it.
We included Depsky in our protection mechanisms section because it presents novel ideas that might be applicable to new schemes trying to add security to the cloud. In this solution we do not required to trust only in an administrator from one of the clouds, instead we assume that there is not going to be a collusion attack from administrator of all the involved clouds. This assumption seems to hold better.

Chapter 5

Conclusions and Future Work

We state our conclusions for this graduate project before moving on to discuss some possible tracks for further investigation.

5.1 Conclusions

Cloud computing is making its way into the information technology industry [6, 7, 8, 9], and at the same time it is capturing a considerable amount of attention within the research community [12, 13, 32, 1, 37, 46, 2, 14]. However, taking into account the attacks discussed in this document, it is safe to say that there is still quite an amount of work to be done if we want to achieve holistic security in current Cloud computing solutions. Like we have discussed, from a malicious administrator's perspective, it is still possible to compromise confidential data that belongs to the Cloud user, e.g., passwords, private keys or intellectual property stored in the Cloud infrastructure. The security deficiencies causing these problems do not have a linear solution that can be readily applied to current Cloud infrastructures in order to provide an immediate resolution to existing issues. It is also important to notice that the attacks demonstrated in this thesis raise a level of concern greater than the attack described in [29] because, in our setting, we do not even need to reboot the machine. We can easily obtain the required memory dump without being noticed.
We have seen that the problems surrounding the implementation of an integrity-protected hypervisor are still being actively discussed in the research community. The problems raising the greatest concern are the inability of current formal methods not being able to prove the correctness of considerably complex systems [51]. This fact has a negative impact on the ability to attest to the integrity of the trusted computing base [13, 12] required as a foundation to build a trustworthy Cloud computing infrastructure. This clearly indicates that

the security of Cloud computing solutions is inversely proportional to its complexity. From this statement, we can infer that platform and software as a service solutions providers have a long journey ahead of them if they want to guarantee to their users that in no way their privacy is being violated when they use their solutions. We are not excluding infrastructure as a service from the group, but since it is the least complex Cloud computing service model, it can be the one with an easier path to overcoming its security issues.
Besides all these arguments there is still the question of how easily an administrator can obtain a memory dump from a machine to which it has physical access as it is exposed in [29]. Therefore, Cloud computing security is never going to be singularly a technology problem. The policies employed in the infrastructures are also going to have an important role to play in achieving security in such environments.

5.2 Future Work

Regarding the problems that were raised in this work we consider that it would be useful to perform the same attacks in commercial cloud solution (e.g., [53]) to check how prepared are those solutions for these kind of attacks. Another interesting work would involve consulting current web hosting providers to learn how are they securing the information that clients willingly deploy in their servers to be available to the Internet user.
In order to find ways to solve the existing issues, we believe that exploring novel protection mechanisms in a way similar to the ones presented by DepSky might be an interesting path to explore. Another important problem to solve is the attestation problem related to VM mobility.

Appendix A

Setting up the environment

In order to perform tests that could help us in determining the level of existing privacy in Cloud Computing, we have setup an environment using the latest version of the Xen Hypervisor, with Ubuntu 10.04 Server LTS as Dom-0 operating system. This section intends to provide a guide to how anyone can easily assemble this environment and perform the same tests we did. We are going to give you a step-by-step guide for every stage of the configuration. However, the reader has to bear in mind that this guide is written in accordance with current versions. If you have different versions try to follow our explanation and at the same time solve any new questions that come along.

A.1 Building the Dom-0 capable Linux kernel

To begin, we first downloaded and installed Ubuntu 10.04 Server LTS. Then we used it has a base system to build the Dom-0 capable kernel. To build the aforementioned kernel the following steps were needed.
First we install some required tools and libraries.
root@host:/home/root# apt-get install libcurl4-openssl-dev \
xserver-xorg-dev mercurial gitk build-essential libncurses5-dev \
uuid-dev gawk gettext texinfo bcc libncurses5-dev dpkg-dev \
debhelper iasl texinfo bridge-utils bison flex

root@host:/home/root# apt-get build-dep xen-3.3

Our next step is cloning the git repository where developers keep current and under development Xen kernel source.

root@host:/home/root# cd /usr/src

root@host:/usr/src# git clone git://git.kernel.org/pub/scm/linux/kernel/git/jeremy/xen linux-xen

root@host:/usr/src# cd linux-xen

root@host:/usr/src/linux-xen# git checkout –b xen/stable-2.6.32.x origin/xen/stable-2.6.32.x

At this stage we already have a clone of the current git Xen kernel repository and we can proceed to build the kernel. The first step is very important because it is where we are going to enable Dom-0 and Xen support. In order to do so we run the following command and make sure the right options are selected before calling make.

root@host:/usr/src/linux-xen# make menuconfig

After running this command we need to select the following options in the configuration menu or edit the .config file and make sure they all have the right configuration value.

CONFIG_HIGHPTE=n # For 32 bit PAE version
CONFIG_ACPI_PROCFS=y
CONFIG_XEN=y
CONFIG_XEN_MAX_DOMAIN_MEMORY=32
CONFIG_XEN_SAVE_RESTORE=y
CONFIG_XEN_DOM0=y
CONFIG_XEN_PRIVILEGED_GUEST=y
CONFIG_XEN_PCI=y CONFIG_PCI_XEN=y
CONFIG_XEN_BLKDEV_FRONTEND=m
CONFIG_NETXEN_NIC=m
CONFIG_XEN_NETDEV_FRONTEND=m
CONFIG_XEN_KBDDEV_FRONTEND=m
CONFIG_HVC_XEN=y
CONFIG_XEN_FBDEV_FRONTEND=m
CONFIG_XEN_BALLOON=y
CONFIG_XEN_SCRUB_PAGES=y
CONFIG_XEN_DEV_EVTCHN=y
CONFIG_XEN_BACKEND=y
CONFIG_XEN_BLKDEV_BACKEND=y
CONFIG_XEN_NETDEV_BACKEND=y
CONFIG_XENFS=y
CONFIG_XEN_COMPAT_XENFS=y
CONFIG_XEN_XENBUS_FRONTEND=m
CONFIG_XEN_PCIDEV_FRONTEND=y

Finally, we issue the commands that are going to finish building the kernel. When these commands finish we have built the kernel and it is ready to use.

root@host:/usr/src/linux-xen# make –j12

root@host:/usr/src/linux-xen# make modules_install install

root@host:/usr/src/linux-xen# chmod g-s /usr/src –R

root@host:/usr/src/linux-xen# make deb-pkg

root@host:/usr/src/linux-xen# dpkg –i ../linux-image*<kernel_version>.deb

root@host:/usr/src/linux-xen# depmod <kernel_version>

root@host:/usr/src/linux-xen# update-initramfs –c –k <kernel_version>

A.2 Building the Xen Hypervisor

The next set is to build the hypervisor we are going to use with the kernel we have just built. In our case, we are going to clone Xen's repository in order to obtain their last stable version. The following commands are the ones we need to complete the process.

root@host:/usr/src# hg clone http://xenbits.xensource.com/xen-4.0-testing.hg

root@host:/usr/src# cd xen-4.0-testing.hg

root@host:/usr/src/xen-4.0-testing.hg# make xen tools stubdom

root@host:/usr/src/xen-4.0-testing.hg# make install-xen install-tools PYTHON_PREFIX_ARG= install-stubdom

We want xend and xendomains to run as services so we need to update the list of services running at startup with the following commands.

root@host:/usr/src# update-rc.d xend defaults 20 21

root@host:/usr/src# update-rc.d xendomains defaults 21 20

A.3 Update Grub

When the previous points are successfully completed we have finished building the required software to boot into our Dom-0 kernel. We need to add an entry to the configuration file of Grub boot loader. If you are using a different boot loader just perform the same operation for it. Following is the entry we used.

menuentry "Xen 4 / Ubuntu 10.04 Kernel 2.6.32.15"{
set root='(main_vol-boot)'
multiboot /xen.gz dummy=dummy dom0_mem=512M
module /vmlinuz-2.6.32.15 dummy=dummy nopat root=/dev/mapper/main_vol-root ro console=tty0
module /initrd.img-2.6.32.15
}

We can now reboot the system and boot into our Xen Dom-0 enabled kernel. All subsequent steps are performed in the new Dom-0 environment we have just setup.

A.4 Setting up the unprivileged domain

We now have an up and running privileged domain (Dom-0) to which we can issue the necessary commands to configure an unprivileged domain (Dom-U). This unprivileged domain is going to be the victim system. The machine where we built our Dom-0 environment supports HVM and we are using this capability to setup our victim Dom-U. If your machine does not support HVM you can setup a paravirtualized Dom-U. Our configuration file contains the following options.

kernel="/usr/lib/xen/boot/hvmloader"
builder="hvm"
memory=512
name="LucidSRV"
vcpus=1
vif=['type=ioemu, bridge=eth0']
disk=['phy:/dev/mapper/main_vol-domu,sda,w', 'file:/ubuntu.iso,hda:cdrom,r']
disk=['phy:/dev/mapper/main_vol-domu,xvda,w']
device_model='/usr/lib/xen/bin/qemu-dm'
vnc=1
vnclisten="0.0.0.0"
vncpasswd='12345'
stdvga=0
boot='d'
usbdevice='tablet'

The lines in red are only to be used in the first boot of the Dom-U. Those two lines are used to boot the ISO file from which we are going to install the Ubuntu distribution into the volume created for the Dom-U. Our volume was created using LVM, which is currently

widely used. We then use xend manager (xm) to create the unprivileged domain as follows.

root@host:# xm create domu_config_file.cfg

After this command finishes with no errors we can use our favorite VNC client and connect to the virtual machine we have just created to proceed with the Ubuntu installation as we normally would in a unique physical machine. In this step, we chose to install a LAMP solution so we can use apache as an example in our tests. When the installation finishes we remove the machine, comment the red lines and create the unprivileged domain again. This time the system will boot into our fresh installation and is ready to use.

A.5 Setup SSL in Apache

We decided to setup SSL in our Apache web server in order to perform an attack where we try to acquire the RSA private key used to establish the secure SSL connection with the web server configured and running in the unprivileged domain we have just created. To do so, I now explain you the required steps to create a Certificate Authority (CA) and then sign a certificate for our web server using our own CA. Here we are not worried with the legitimacy of the certificate we use, so there is no problem in creating our own CA to sign a certificate. We are not suggesting that you should setup a commercial website using a scheme such as this, it is not recommended. For commercial websites, the user should request a certificate from a well-known and trusted CA. The only purpose of this certificate is to be attacked.
The first step is to create a private key and a private CA X.509 certificate for our homemade CA. You can create a directory to hold this confidential data.

root@host:# mkdir /home/<user>/CA

root@host:# chmod 0770 /home/<user>/CA

root@host:# cd /home/<user>/CA

root@host:/home/<user>/CA# openssl genrsa –des3 –out ca-key.key 1024

The previous command generates a 1024-bit private RSA key encrypted with DES for the CA. The subsequent command creates a certificate, valid for a year, using the key openssl just generated for us. When you issue this command you will need to fill-in some information regarding the CA, just insert some fictional information.

root@host:/home/<user>/CA# openssl req –new –x509 –days 365 –key ca-key.key –out ca-certificate.crt

We have now our own CA we can proceed to create the key and certificate for our secure Apache web server. The next three commands are what we require to achieve that purpose. In the second command you need to insert the right name of the host machine. If you do not have a domain use the IP address.

root@host:/home/<user>/CA# openssl genrsa –des3 –out apache-server.key 1024

root@host:/home/<user>/CA# openssl req –new –key apache-server.key –out apache-server.csr

root@host:/home/<user>/CA# openssl –req –in apache-server.csr –out apache-server.crt –sha1 –CA ca-certificate.crt –CAkey ca-key.key –CAcreateserial –days 365

root@host:/home/<user>/CA# chmod 0400 *.key

This set of commands finishes the phase of generating the required certification materials for the creation of a secure SSL Apache web server. The subsequent step is to configure Apache to use the certificate and key we have just generated for it. The set of commands that follow are what we need to reach that outcome.

(a) copy the certificate and key files to an appropriately named directory

root@host:/home/<user>/CA# cp apache-server.key /etc/apache2/ssl

root@host:/home/<user>/CA# cp apache-server.crt /etc/apache2/ssl

(b) enable SSL

root@host:/home/<user>/CA# a2enmod ssl

(c) create the SSL .conf file and establish a necessary link with the enabled directory

root@host:/home/<user>/CA# cp /etc/apache2/sites-available/default /etc/apache2/sites-available/ssl

root@host:/home/<user>/CA# ln –s /etc/apache2/sites-available/ssl /etc/apache2/sites-enabled/ssl

(d) set up different document roots for clear html and html over SSL

root@host:/home/<user>/CA# mkdir –p /var/www/html

root@host:/home/<user>/CA# mkdir –p /var/www-ssl/html

(e) configure the document roots and ports accordingly

root@host:/home/<user>/CA# vim /etc/apache2/sites-available/default

In this file we need to make sure we use port 80 and the /var/www/html document root.

root@host:/home/<user>/CA# vim /etc/apache2/sites-available/ssl

This second file needs to use port 443 and /var/www-ssl/html as document root. We also need to turn one the SSL engine by adding the following lines:

SSLEngine On

SSLCertificateFile /etc/apache2/ssl/apache-server.crt

SSLCertificateKeyFile /etc/apache2/ssl/apache-server.key

(f) restart apache web server

root@host:/home/<user>/CA# /etc/init.d/apache2 restart

If everything was done correctly we now have a secure Apache Web server configured and running in our unprivileged domain. We can move to our security tests.

Bibliography

[1] Sun Microsystems Inc, Take your business to a higher level, Sun Cloud Computing, March 2009. 1, 2.1, 5.1

[2] M., Armbrust, et al. Above the clouds: A Berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, U.C. Berkeley, Feb 2009. 1, 2.1, 5.1

[3] P. Mell and T. Grance, The NIST definition of Cloud Computing, NIST, July 2009. 2.1

[4] Apache Hadoop, http://hadoop.apache.org/, September 2010. 2.1

[5] MogileFS, http://danga.com/mogilefs/, September 2010. 2.1

[6] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/, September 2010. 2.1.1, 5.1

[7] Windows Azure Platform, http://www.microsoft.com/windowsazure/getstarted/, September 2010. 2.1.2, 5.1

[8] Google App Engine, http://code.google.com/intl/pt-PT/appengine/, September 2010. 2.1.2, 2.1.3, 5.1

[9] Salesforce.com, http://www.salesforce.com/eu/crm/, September 2010. 2.1.3, 5.1

[10] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third- party compute clouds. In Proceedings of the ACM Conference on Computer and Communications Security, 2009. 2.2.1, 4.2.3, 4.3.3.4

[11] M. van Dijk and A. Juels, On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Computing, In the Proceedings of the 5th USENIX Workshop on Hot Topics in Security, August 2010. 2.2.2

[12] J. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, TrustVisor: Efficient TCB Reduction and Attestation, In Proceedings of the IEEE Symposium on Security and Privacy, May 2010. 2.2.3, 3.4, 4.1.2, 4.1.3, 4.2.3, 5.1

[13] A. Vasudevan , J. M. McCune, N. Qu, L. van Doorn and A. Perrig, Requirements for an Integrity-Protected Hypervisor on the x86 Hardware Virtualized Architecture. In Proceedings of the 3rd International Conference on Trust and Trustworthy Computing, June 2010. 2.2.3, 3.4, 3.6, 4.3.3.1, 5.1

[14] D. G. Murray, G. Milos, and S. Hand. Improving Xen security through disaggregation. In Proceedings of the ACM Virtual Execution Environments, pages 151–160, March 2008. 2.2.3, 3.4, 3.4, 4.1.3, 4.2.3, 5.1

[15] M. P. Correia and P. Sousa, Segurança no Software, FCA editores. September 2010. 2.3, 2.3.2, 2.3.2.2

[16] Xen Hypervisor, http://www.xen.org/, September 2010. 2.3.1

[17] VMWare Workstation, https://www.vmware.com/products/workstation/, September 2010. 2.3.1

[18] The Honeynet Project, http://www.honeynet.org/, September 2010. 2.3.2.2

[19] Xen.org, What is Xen Hypervisor?, http://www.xen.org/files/Marketing/WhatisXen.pdf, September 2010. 2.3.3

[20] Trusted Computing Group (TCG), http://www.trustedcomputinggroup.org/, September 2010. 2.4, 3.4

[21] A. Martin, The ten-page introduction to Trusted Computing, Technical Report CS-RR-08-11 - Oxford University Computing Laboratory, November 2008. 2.4, 2.4.1

[22] D. Grawrock, The Intel Safer Computing Initiative Building Blocks for Trusted Computing, Intel Press, 2006. 2.4, 2.4.1, 2.4.4

[23] Trusted Computing Group - FAQs, http://www.trustedcomputinggroup.org/developers/trusted_platform_module/faq, September 2010. 2.4.1

[24] E. Brickell, J. Camenisch, and L. Chen: Direct anonymous attestation. In Proceedings of 11th ACM Conference on Computer and Communications Security, 2004. 2.4.5

[25] Direct Anonymous Attestation, http://www.zurich.ibm.com/security/daa/, IBM Zurich Research Lab, 2004. 2.4.5, 4.3.3.3

[26] Sherri Davidoff, Cleartext Passwords in Linux Memory, http://philosecurity.org/pubs/davidoff-clearmem-linux.pdf, MIT, July, 2008. 3.1, 3.1, 3.1

[27] TrueCrypt, http://www.truecrypt.org/, September 2010. 3.1, 3.1

[28] Cold Boot attack, http://citp.princeton.edu/memory/, July 2008. 3.2, 3.2

[29] E. W. Felten et al, Lest We Remember: Cold Boot Attacks on Encryption Keys, 17th USENIX Security Symposium, San Jose, CA, USA 2008. 1, 3.1, 3.2, 3.2, 4.3.3.2, 5.1

[30] RSA LABORATORIES. PKCS #1 v2.1: RSA cryptography standard. ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf. 3.2

[31] Dark Throne, http://www.darkthrone.com/, Lazarus Software, September 2010. 3.3

[32] N. Santos, K. P. Gummadi, and Rodrigo Rodrigues, Towards Trusted Cloud Computing, In Proceedings of the Workshop on Hot Topics in Cloud Computing, 2009. 3.4, 4, 4.3.1, 5.1

[33] D. Robbins, Common threads: Learning Linux LVM, Part 1, http://www.ibm.com/developerworks/linux/library/l-lvm/index.html, 2001. 3.3

[34] PrivacyCA, http://www.privacyca.com/, September 2010. 3.4, 3.4

[35] S. Berger et al., vTPM: Virtualizing the Trusted Platform Module, In Proceedings of the 15 th USENIX Security Symposium, 2006. 4, 4.1.1, 4.2.2

[36] S. Curry et al. Infrastructure security: Getting to the bottom of compliance in the cloud, March 2010. RSA Security Brief. 4.1.3

[37] F. John Krautheim, Private Virtual Infrastructure for Cloud Computing, In Proceedings of the Workshop on Hot Topics in Cloud Computing, 2009. 4.2.1, 5.1

[38] The Coronor's Toolkit, http://www.porcupine.org/forensics/tct.html, September 2010. 3.1

[39] Xen Documentation, xen-4.0.1/docs/misc/dump-core-format.txt, September 2010. 3.1

[40] William Stallings, Network Security Essentials: Applications and standards, 2nd Edition, Prentice Hall Professional Technical Reference, 2002. 3.2

[41] Public-Key Cryptography Standard (PKCS) #12, http://www.rsa.com/rsalabs/node.asp?id=2138, RSA Labs, September 2010. 3.2

[42] ASN.1 Project, http://www.itu.int/ITU-T/asn1/introduction/index.htm, International Telecommunication Union, September 2010. 3.2

[43] X.690 : Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), http://www.itu.int/rec/T-REC-X.690-200811-I, International Telecommunication Union, September 2010. 3.2

[44] X.680 : Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation, http://www.itu.int/rec/T-REC-X.680-200811-I, International Telecommunication Union, September 2010. 3.2

[45] Wuala - Secure Online Storage - Backup. Store. Share., http://www.wuala.com/en/, LaCie, Caleido AG, September 2010. 3.3

[46] Lionel Litty, H. Andrés Lagar-Cavilla, and David Lie, Computer Meteorology: Monitoring Compute Clouds, In Proceedings of the 12th Workshop on Hot Topics in Operating Systems, 2009. 3.5, 3.5, 5.1

[47] Lionel Litty, H. Andrés Lagar-Cavilla, and David Lie, Hypervisor Support for Identifying Covertly Executing Binaries, Proceedings of the 17th conference on Security symposium pp. 243-258, 2008. (document), 3.12, 3.5

[48] TCG Software Stack (TSS) Specification, http://www.trustedcomputinggroup.org/ resources/ tcg_software_stack_tss_specification, Trusted Computing Group, September 2010.

[49] David Challener, Kent Yoder, Ryan Catherman, David Safford, Leendert Van Doorn, A Practical Guide to Trusted Computing, IBM Press, 2008. 2.4.6

[50] TrouSerS - The open-source TCG Software Stack, http://trousers.sourceforge.net/, September 2010. 2.4.6

[51] Jonathan M. McCune, Reducing the Trusted Computing Base for Applications on Commodity Systems, PhD Thesis, School of Electronic and Computer Engineering, Carnegie Mellon University, January 2009. 5.1

[52] B. Quaresma, A. Bessani and P. Sousa, Melhorando a Fiabilidade e Segurança do Armazenamento em Clouds, In Actas do INFORUM 2010 - Simpósio de Informática, September 2010. 4.4.1

[53] VMWare Cloud Computing solutions, http://www.vmware.com/solutions/cloud-computing/, October 2010. 5.2

[54] D. Parkhill, The Challenge of the Computer Utility, Addison-Wesley Publishing Company, 1966. 2.1

[55] A. Miyaji and K. Umeda. A fully-functional group signature scheme over only known-order group. LNCS, 3089:164-179, 2004. 2.4.5

[56] National Computer Security Center, "Trusted Computer Systems Evaluation Criteria, DoD Computer Security Center, CSC-STD-001-83, August 1983. 2.4.1

[57] Linux Logical Volume Manager, http://www.linuxconfig.org/images/Lvm.jpg, October 2010. (document), 3.7

[58] Attack from Subsection 3.1: Clear text passwords in Linux memory dump, http://www.youtube.com/watch?v=FJryJ3gYSkc, October 2010. 3.1

[59] Attack from Subsection 3.2: Obtaining private keys using memory snapshots, http://www.youtube.com/watch?v=8xKhS5ZGR5s, October 2010. 3.2

[60] Attack from Subsection 3.3: Extracting private data from the hard disk, http://www.youtube.com/watch?v=wvlnIA49spY, October 2010. 3.3

[61] Attack from Subsection 3.4: Virtual Machine Relocation, http://www.youtube.com/watch?v=Z9o9-pAtSp0, October 2010.

3.4